BIG DATA

HADOOP

THE BEGINING

BY

Mr SREERAM

Data Scientist

Ventech IT Solutions

307/B Neelgiri block, Adithya enclave,
Ameerpet, Hyderabad-38

PH: 040-65356678, 9030056678

శ్రీరామ ఓం నమః శివాయ

By Mr. SREERAM (Data Scientist)

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678

Hadoop

Hadoop is a big data framework to store and process high volumes of data with very high speed.

What is Big Data?

Big Data is a large collections of Huge volume data sets with complex data structures.

Symply,

Big Data = Huge Volume + Big Complex data

examples for Complex data is Unstructured data, grapher data (ex:- Facebook friend connections etc) weblogs & database logs etc...

problems of RDBMS for dealing Big Data

(i) As data volume increases speed decreasing

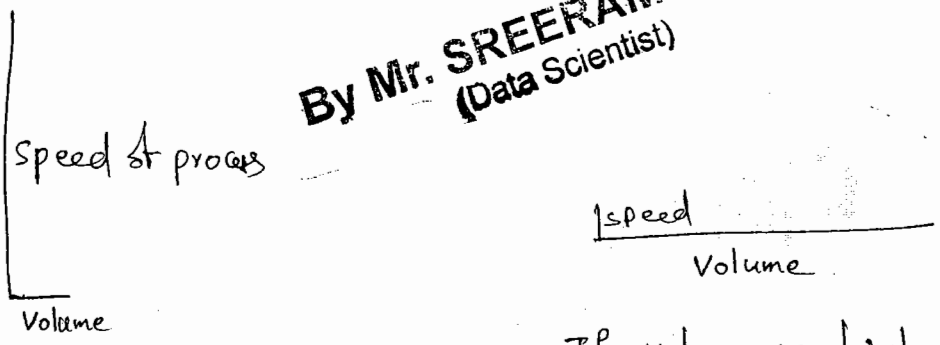
(ii) As complexity of algorithms increasing, speed will be decreasing

(iii) RDBMS can not process ~~unt~~ unstructured data.

Problems of RDBMS

(1) As data volume increasing, speed will be decreasing

ex:-



If Volume less, then Speed is High

If Volume is high, Speed will be down.

ex:-

a) select sum(amt) from sales; → 1000 Rows

b) select sum(amt) from sales2; → 1 Lakh rows

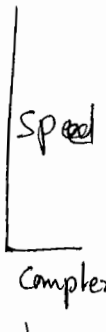
c) select sum(amt) from sales3; → 1 Cr rows.

from all above sql queries,
first query execution is very faster.
Suppose if first query requires 1 min time,
2nd may take 100 mins of process time,
3rd may take hours of time.

Problems of RDBMS

(ii) As complexity of algorithm is increasing, speed will be decreasing

ex:-



If complexity is less, then high speed

$\frac{1}{\text{complexity}}$ speed

If complexity is high, then speed is very low, eventhough volume is less.

ex:-

a) select sum(amt) from sales;
b) select avg(amt) from sales;
c) select stddev(amt) from sales;

→ 1000 Rows (Volume same in each table)

In above queries,
first one is very faster.
2nd will take little extra time, when compared with 1st one.
because to calculate avg, Sum to be processed seperately and count to be processed seperately and again avg = sum/count to be processed.
But when you take 3rd query (stddev) is, standard deviation which is statistical fundion, lots of internal proces to be done, so it will take more time when compared with first and 2nd query.

So, finally we can understand the RDBMS speed will be controlled by two things

(i) Volume (data)
(ii) Complexity of Algorithms.

especially In case of analytics, data science fields we need to process lots of complicated algorithms to process and analyze data..

example algorithms

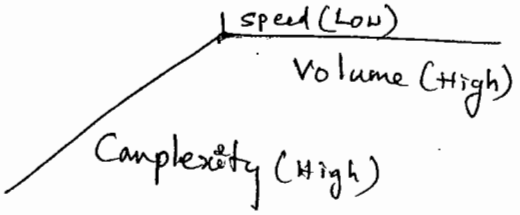
→ Decision trees
→ Random forests
→ FPGrowth
→ Collaberative filtering - etc.

for these algorithms, database will take lots of processing time, eventhough data volume is less.

Just think,

what if, if such kind of complicated Algorithms applied on Terabytes of data & peta bytes of data.

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678



Problems of RDBMS

(iii) All varieties of data process/Analysis is not possible.

→ RDBMS is good for structured data processing

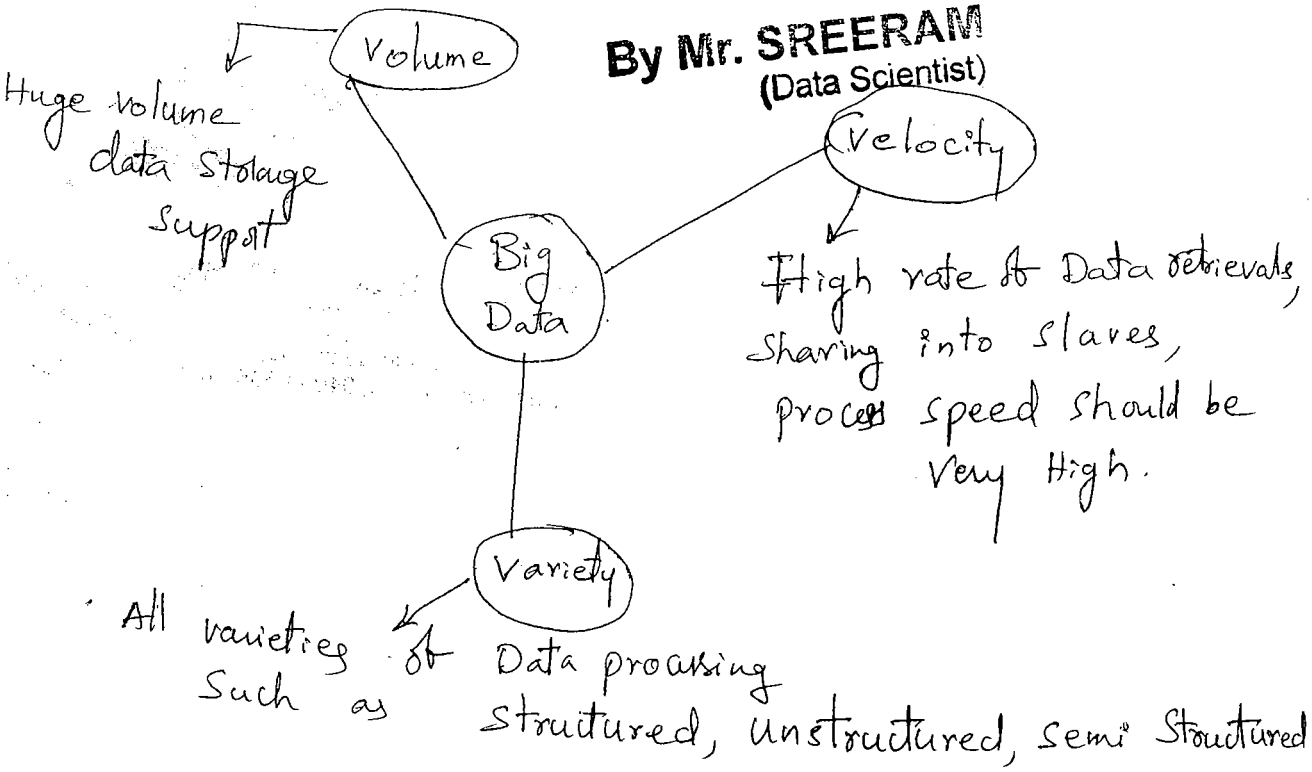
→ It can not analyze unstructured text, Images, audios, videos etc.

→ Now a days RDBMS databases can process Semi structured data, such as XML, json etc.. but not all complications of XML and json.

BIG DATA DIMENSIONS (CHARACTERISTICS)

To call some SW/Data processing System as BigData framework, the System should satisfy dimensions of BigData.

Big Data Dimensions



Hadoop satisfies all characteristics of BigData.

so you can say hadoop as a bigdata framework

Other BigData frameworks in Market.

→ SAP HANA

→ Oracle Exadata .. etc.

But these are not satifying all dimensions of BigData.

→ Unlimited data volumes like multiple petabytes is not possible.

→ with in limits, speed is good ✓(velocity) when canpared with traditional data-base systems

→ different varieties of data process not possible.

HANA and Exadata also can proces only structured.

Hadoop Advantages.

Technical Benifits:-

1) Unlimited data storage

2) Very Very High Speed
(ex:- 1 Terabyte Sort in 3 minits)

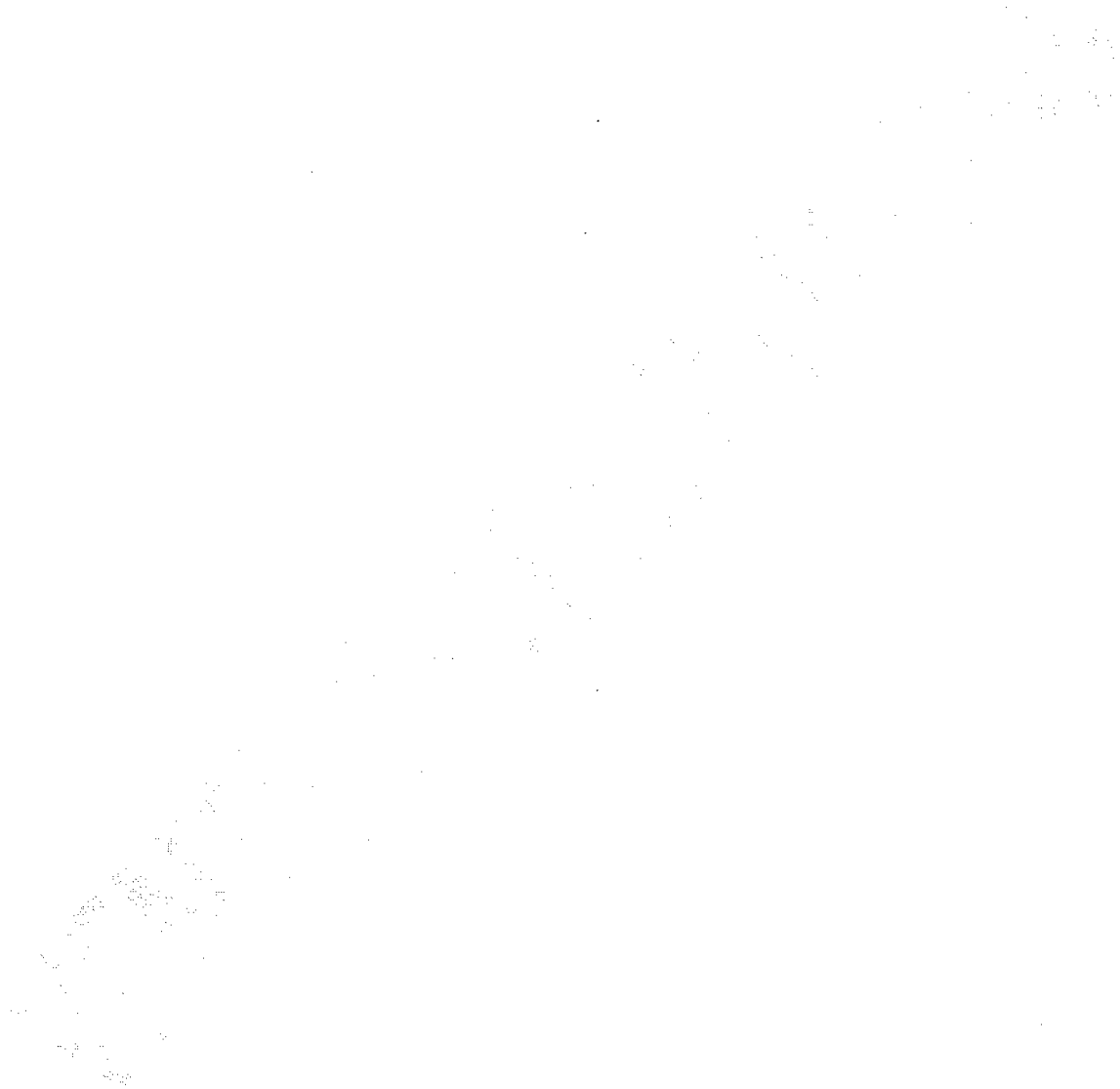
3) All Varieties of data process
Such as Structured → table data, delimited files ...
Unstructured → Text, Images, Audios, Videos etc.
Semi Structured → XML, JSON, URLs ...

Financial Benifits (Budjet) of Hadoop

1) is open source System (SW)
   → is a product of Apache
   → No Licence Fee.

2) Hadoop can be deployed on Commodity Hardware
   → If you take teradata, HANA, Netezza such systems are very High end Hardwares.
   → teradata, Netezza these systems are appliences.
   ex:-
   → Hardware given by teradata corp (Teradata boxes)
   → SW (Licenced - costly) given by teradata.
   → In Appliences SW comes along with HW.
   → Very High investment.

→ But you can deploy hadoop on normal CPUs (HW)
ex:-
→ you can implement 20TB space for your datawarehouse in just 5 lakhs (< 5Lak)
→ if Something in teradata, it may be 2 Cr to 3 Cr of investment range.



Hadoop has two services

(i) Storage → for Storage HDFS file System.

(ii) process → MapReduce.

↳ is an execution model in hadoop framework.

HDFS has two advantages,

(i) Unlimited data Storage

(ii) It Supports Parallel process.

Server Scaling models:

(i) Vertical Model

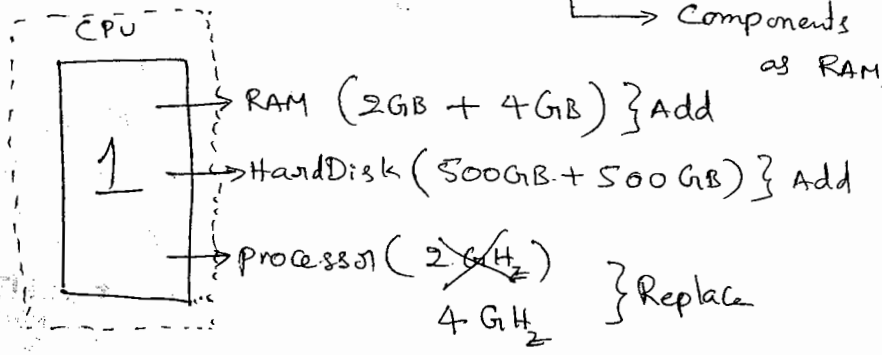
(ii) Horizantal Model

(i) Vertical Model :-

→ Server has only 1 CPU.

→ By Adding/Replacing components, Server scaleup is done.

↳ Components such as RAM, HD, Processor..



→ Problem of vertical model is high level scaling is not possible.

ex:- you can not make your Laptop RAM as 1TB.

because, 1TB RAM not available,

even available, the processor may not support.

→ Availability is one problem

→ Compatability is another problem.

(ii) Horizantal Model

To scaleup server capacity in high level, Horizantal model is introduced with distributed style.

→ By Adding CPUs to CLUSTER, we scaleup capacity.

→ CLUSTER act as a Server

→ CLUSTER :-

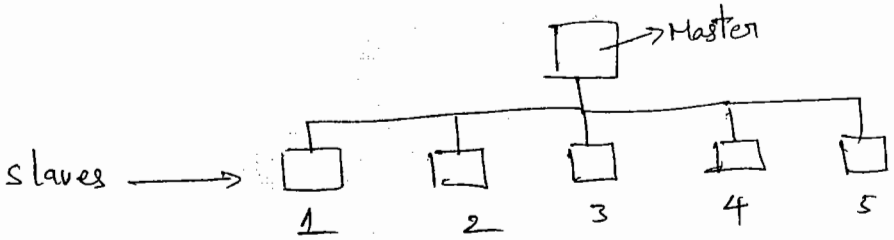
A group of network of CPUs act as a server.

→ In CLUSTER, each CPU called as a NODE.

→ CLUSTER has two types of nodes.

(i) Master node

(ii) Slave node



Master →

→ Work assignments
→ Load balancing
→ Fault Tolerance
→ Health Monitoring
⋮

} High level operations

Slaves → Data Stored in slaves

→ Process (Task executions) by slaves } Slavery work.

Example 1 — WordCount.

```
$ cat > comment
I love India
India love you
you love India
```

```
$ hadoop fs -mkdir mrdemo
```

```
$ hadoop fs -copyFromLocal comment mrdemo
```

now we need each word occurances.

example o/p.

I	1
love	3
⋮	

WordCount.java


```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapredua.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conb.Configuration;
import org.apache.hadoop.fs.Path;
import java.io.IOException;
import java.util.StringTokenizer;
```

```
public class WordCount
{
  public static class WordMap extends Mapper<LongWritable,
                                  Text, Text, IntWritable>
  {
    public void map(~~Text k~~, LongWritable k,
          Text v, Context con)
      throws IOException, InterruptedException
    {
        String line = v.toString();
        StringTokenizer t = new StringTokenizer(line);
        while(t.hasMoreTokens())
        {
            String word = t.nextToken();
            con.write(new Text(word), ~~new IntWritable(1)~~);
                                      new IntWritable(1));
        }
    } //end of map()

} //end of Mapper class
public static class RedForSum extends Reducer<Text,
                                  IntWritable, Text, IntWritable>
  {
      public void reduce(Text k,
          Iterable<IntWritable> vlist, Context con)
       throws IOException, InterruptedException
      {
          int tot=0;
          for(IntWritable v: vlist)
              tot+= v.get();
          con.write(k, new IntWritable(tot));
      } //end of reduce()
} //end of Reducer class
```

```
public static void main(String[] args)
                    -throws Exception
{
    Configuration c = new Configuration();
    Job j = new Job(c, "WordCount");
    j.setJar
    j.setJarByClass(WordCount.class);
    j.setMapperClass(WordMap.class);
    j.setReducerClass(RedForSum.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);
  Path p1 = new Path(args[0]);
  Path p2 = new Path(args[1]);
  FileInputFormat.addInputPath(j, p1);
  FileOutputFormat.setOutputPath(j, p2);
 System.exit(j.waitForCompletion(true) ? 0:1);
} // end of main()

} // end of Driver class — WordCount
```

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678

→ Export this class onto jar file.

ex:- /home/training/Desktop/mrdemo.jar

How to submit job:-

$ hadoop fs -cat comment

I love India
India love you
you love India

$ hadoop jar /home/training/Desktop/mrdemo.jar \
mr.analytics.WordCount \
~~/home/training/Desktop/mrdem~~
/user/training/Comment \
/user/training/WCResult ↵

→ Output Directory (HDFS)

$ hadoop fs -ls /user/training/WCResult

part-r-00000

$ hadoop fs -cat /user/training/WCResult/part-r-00000

I 1
India 3
love 3
you 2

'r' Indicates o/p is written by Reducer.

If reducer is suspended, you get part-m-000000

→ Mapper

Example-2

```
$ cat > emp
101, Amar, 20000, M, 11
102, Amala, 30000, F, 12
103, Ankit, 40000, M, 12
104, Ankita, 50000, F, 11
105, Anuz, 60000, M, 11
106, Anusha, 70000, F, 12
107, Akash, 80000, M, 12
^d
```



```
$ hadoop fs -copyFromLocal emp modemo
```

Task :- for each sex group total salary required.

o/p Expected,

F	150000
M	220000

equivalant hql statement:

```
hive> select sex, sum(sal) from emp group by sex;
```

In Mapper class,
we need to seperate(write) Sex as key and sal as Value.

In Reducer we perform Sum Aggregation.

In previous example (WordCount) We have written a Driver class, in that we wrote mapper as innerd class and reducer as inner class.

But better practice is write seperate class for Mapper, Reducer, Driver.

EmpSexSal.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;
import ~~org.ap~~ java.io.IOException;

public class EmpSexSal extends Mapper<LongWritable,
                Text, Text, IntWritable>
{ // schema of file → ecode, name, sal, sex, dno

  ~~String line =~~

  public void map(LongWritable k, Text v,
          Context con) throws IOException,
                          InterruptedExceptim
  {
      String line = v.toString();
      String[] w = line.split(",");
      String sex = w[3];
      int sal = Integer.parseInt(w[2]);
      con.write(new Text(sex), new IntWritable(sal));
  }
}
```

o/p Mapper →

```
M   20000
F   30000
```

RedForSum.java

```
package mr.analytics;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import java.io.IOException;

public class RedForSum extends Reducer<Text, IntWritable,
                                       Text, IntWritable>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
                       Context con) throws IOException,
                                           InterruptedException
    {
        int tot = 0;
        for (IntWritable v : vlist)
        {
            tot += v.get();
        }
        con.write(new Text(
        Con.write(k, new IntWritable(tot));
    }
}
```

o/p →

F	150000
M	220000

Stored in HDFS, then Mapper o/p will be deleted.

How to Execute, already described in example 1.

Driver ⟶ EmpSexSalSum.java


```
pub
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class EmpSexSalSum
{
    public static void main(String[] args)
                throws Exception
    {
        Configuration c = new Configuration();
        Job j = new Job(c,"MyJob");
        j.setJarByClass(EmpSexSalSum.class);
        j.setMapperClass(EmpSexSal.class);
        j.setReducerClass(RedForSum.class);
        j.setOutputKeyClass(Text.class);
        j.setOutputValueClass(IntWritable.class);
        Path p1 = new Path(args[0]); // Input file
        Path p2 = new Path(args[1]); // o/p Directory
FileInputFormat.addInputPath(j,p1);
FileOutputFormat.setOutputPath(j,p2);
System.exit(j.waitForCompletion(true) ? 0:1);
    }
}
```

Examples

input file : /user/training/~~edureka~~ mrdemo/emp

schema : ecode, ename, esal, sex, dno

delimiter : "," (comma)

Task :- for each dno (depart Number) total salary required.

o/p Expected,

11	130000
12	220000

equivalant hql statement

hive> select dno, sum(esal) from emp group by dno;

In Mapper class,

We need to write dno as key, esal as value.

In Reducer class,

We need to perform SUM aggregation.

Already we have RedForSum.class for SUM Aggregation.

We can use it as Reducer.

→

(Turn page for Mapper class)

EmpDnoSal.java.

```
package mr.analytics;

=== } imports

public class EmpDnoSal extends Mapper <LongWritable,
                    Text, Text, IntWritable>

{
    public void map(LongWritable k, Text v,
        Context con) throws IOException, InterruptedException
    {
        String line = v.toString();
        String[] w = line.split(",");
        String dno = w[4];
        int sal = Integer.parseInt(w[2]);
        con.write(new Text(dno), new IntWritable(sal));
    }
}
```


o/p of the Mapper.

11	20000
12	30000
12	40000
11	50000
11	60000
12	70000
12	80000

Driver → to call EmpDnoSal.class as mapper,
and RedForSum.class as Reducer.

EmpDnoSalSum.java

```
package mr.analytics;

.....} imports (same as previous driver class
                — EmpSexSalSum.java) ~~EmpSexSal.java~~

public class EmpDnoSalSum.
{
   public static void main(String[] args) throws Exception
   {
        =
      j.setJarByClass(EmpDnoSalSum.class);
      j.setMapperClass(EmpDnoSal.class);
      j.setReducerClass(RedForSum.class);
        =
   }
}
```

Example 4

Reducer for Average.

RedForAvg.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class RedForAvg extends Reducer<Text, IntWritable,
                                       Text, IntWritable>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
        Context con) throws IOException, InterruptedException
    {
        int tot = 0;
        int cnt = 0;
        for (IntWritable v : vlist)
        {
            tot += v.get();
            cnt ++;
        }
        int avg = tot/cnt;
        con.write(k, new IntWritable(avg));
    }
}
```

Example 5

Reducer for Maximum Value of each group

RedForMax.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
import or

public class RedForMax extends Reducer<Text, IntWritable,
                                        Text, IntWritable>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
            Context con) throws IOException, InterruptedExcepti
    {
        int max = 0; int cnt=0;
        for(IntWritable v: vlist)
        { cnt++
    if (cnt==1) max = v.get();
            max = Math.max(max, v.get());
        }
        con.write(k, new IntWritable(max));
    }
}
```

Example-6

Reducer for minimum value for each data group.

RedForMin.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
public class RedForMin extends Reducer <Text,
    IntWritable, Text, IntWritable >
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
        Context con) throws IOException, InterruptedException
    {
        int min = 0;
        int cnt = 0;
        for (IntWritable v : vlist)
        {
            cnt++;
        if (cnt == 1)  min = v.get();
          min = Math.min(min, v.get());
        }
        con.write(k, new IntWritable(min));
    }
}
```

Example 7

Reducer for Count Aggregation.

RedForCnt.java

By Mr. SREERAM
(Data Scientist)

```
package mr.analytics;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;

public class RedForCnt extends Reducer<Text,
        IntWritable, Text, IntWritable>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
        Context con) throws IOException, InterruptedException
    {
        int cnt=0;
        for(IntWritable v: vlist)
            cnt++;
        con.write(k, new IntWritable(cnt));
    }
}
```

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678

~~Example-7~~

~~Mapper to write dno~~

Now have, two Mapper classes

(i) EmpSexSal.class

(ii) EmpDnoSal.class

5 Reducer Classes

(i) RedForSum.class

(ii) RedForAvg.class

(iii) RedForMax.class

(iv) RedForMin.class

(v) RedForCnt.class

Already we know how to prepare driver class (Job definition)

→ How to set Mapper class

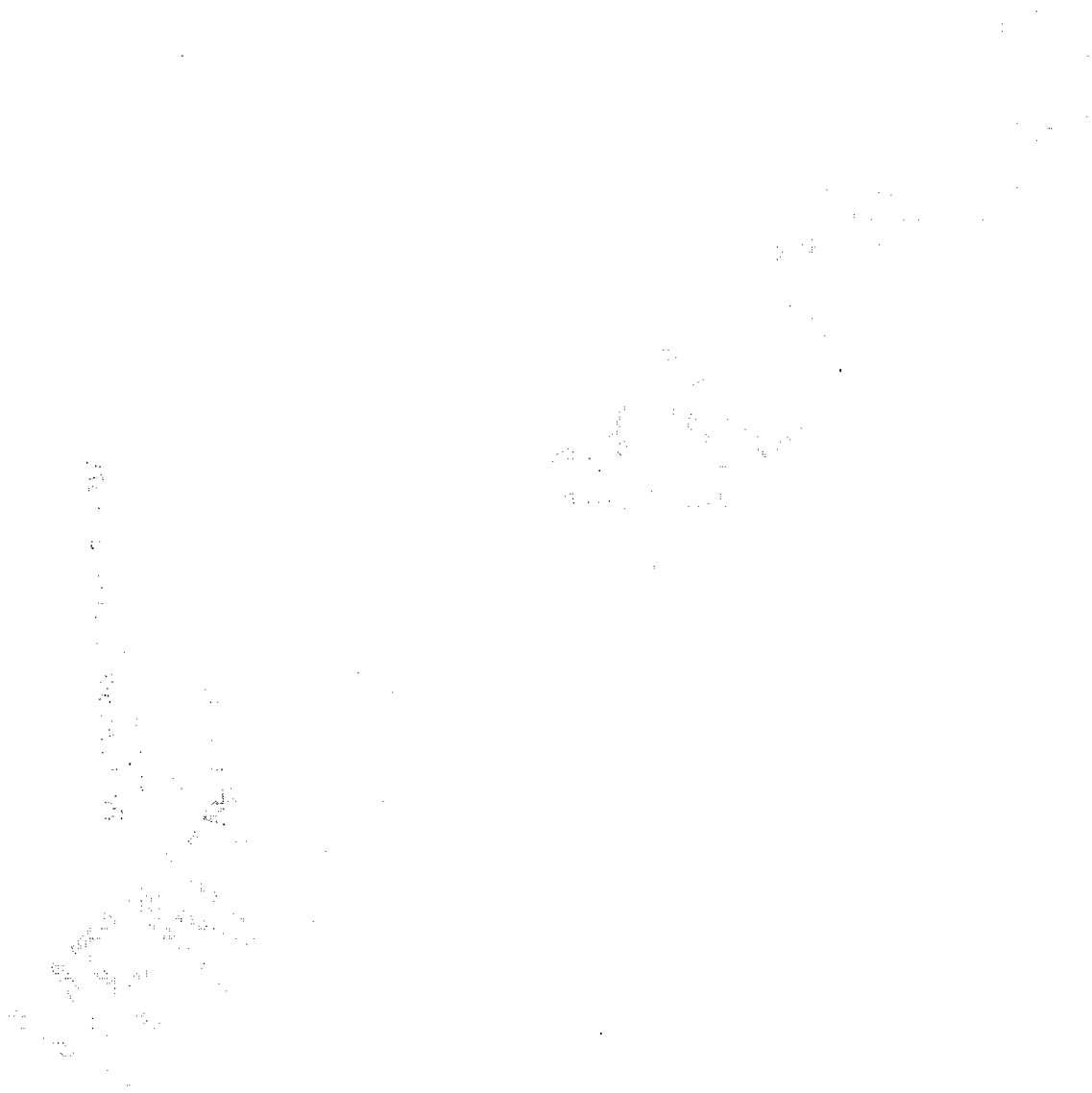
j.setMapperClass(<Mapper name>)

→ How to set Reducer class

j.setReducerClass(<Reducer class>)

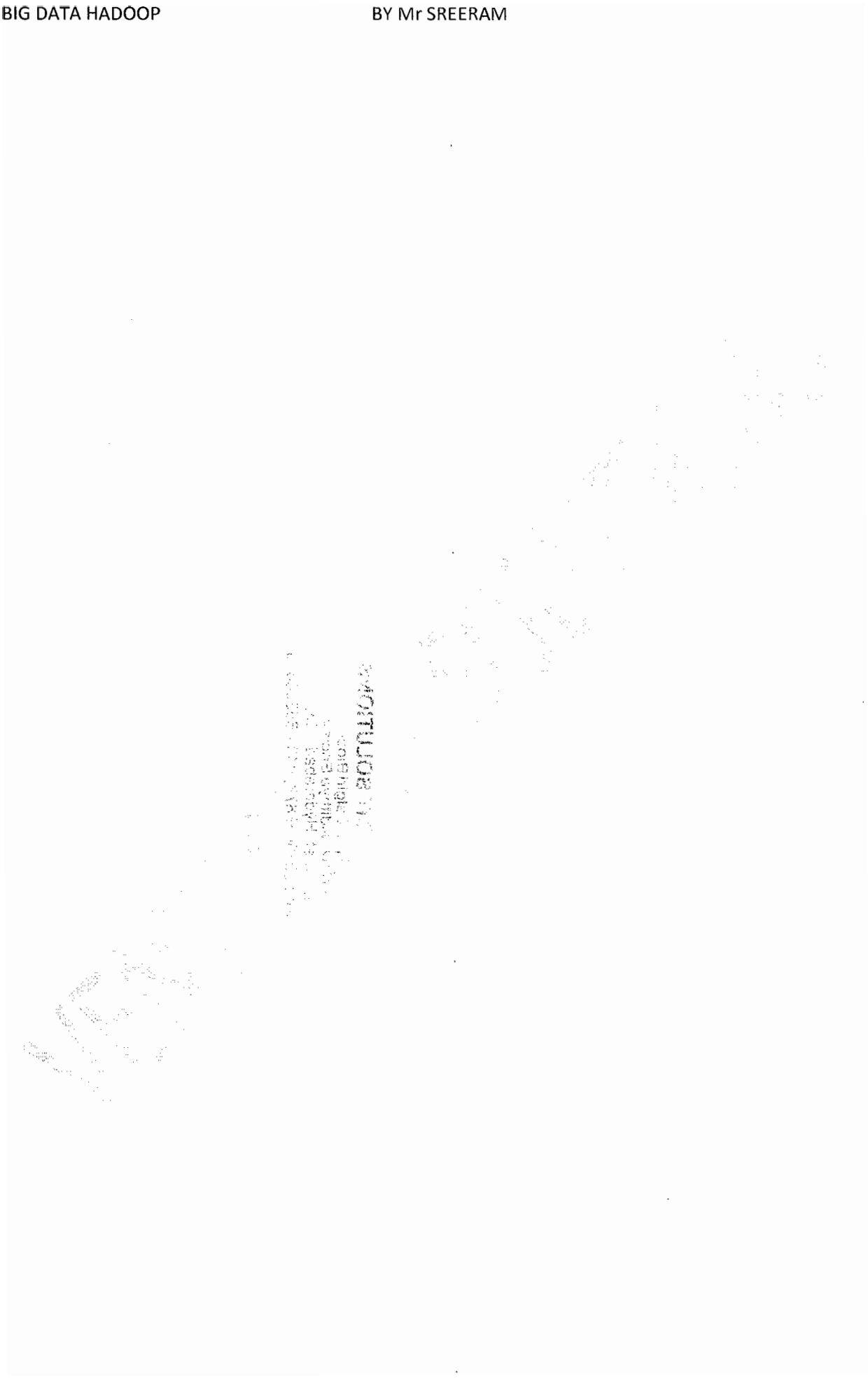
Based on above class we can prepare 10 different Jobs.

Driver class	Mapper class	Reducer class	hql statement
Driver1.class	EmpSexSal.class	RedForSum.class	select sex, sum(sal) from emp group by sex;
Driver2.class	EmpSexSal.class	RedForAvg.class	select sex, avg(sal) from emp group by sex
Driver3.class	EmpSexSal.class	RedForMax.class	select sex, max(sal) from emp group by sex;
Driver4.class	EmpSexSal.class	RedForMin.class	select sex, min(sal) from emp group by sex;



Driver class	Mapper-class	Reducer class	hql statement
Driver5.class	EmpSexSal.class	RedForCnt.class	select sex, count(*) from emp group by sex;
Driver6.class	EmpDnoSal.class	RedForSum.class	select dno, sum(sal) from emp group by dno;
Driver7.class	EmpDnoSal.class	RedForAvg.class	select dno, avg(sal) from emp group by dno;
Driver8.class	EmpDnoSal.class	RedForMax.class	select dno, max(sal) from emp group by dno;

Driver class	Mapper class	Reducer class	hql statement
Driver9.class	EmpDnoSal.class	RedForMin.class	select dno, min(sal) from emp group by dno;
Driver10.class	EmpDnoSal.class	RedForCnt.class	select dno, count(*) from emp group by dno;



In all previous examples,
we performed grouping aggregations based on single key like grouping by sex, & grouping by dno .. etc.

Then how to perform multi grouping?

ex:-

~~hive> select dno, Sum(~~

hive> select dno, sex, Sum(sal) from emp
group by dno, sex;

In this statement,
dno is primary group
it's sub group is sex.

ex o/p:-

11	F	270000
11	M	340000
12	F	—
12	M	—
13	F	—
13	M	—
⋮		

But we can write only one key and one value.

Here technique is,

first seperate dno and sex

Later concatinate it as a single key.

ex:-

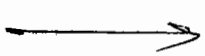
```
String mykey = dno + "\t" + sex;
```

In this way any number of grouping keys we can give.

Example : 8

EmpDnoSexSal.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;
import ~~org.a~~ java.io.IOException;
```



```
/*
   Input file Schema -> ecode, name, sal, ~~dno~~, sex, dno
*/
public class EmpDnoSexSal ~~xxxxx~~ extends Mapper<
   LongWritable, Text, Text, IntWritable >
{
   public void map(LongWritable k, Text v,
   Context con).throws IOException, InterruptedException
   {
      String line = v.toString();
      String[] w = line.split(",");
      String dno = w[4];
      String sex = w[3];
      int sal = Integer.parseInt(w[2]);
      String mykey = dno + "\t" + sex;
      con.write(new Text(mykey), new IntWritable(sal));
   }
}
```

Mapper o/p. (sample)

11	M	25000
11	F	30000
11	M	27000
12	M	30000
13	F	40000
11	F	50000
o/p key		o/p value

Now the task is

for each dno, its sub group sex, total salary is required.

~~Drive~~ We need to call : EmpDnoSexSal.class (Mapper)
and RedForSum.class (Reducer)

Driver → EmpDnoSexSalSum.java

```
package mr.analytics
= } imports
public class EmpDnoSexSalSum
{
    public static void main (String[] args) throws
                                              Exception
    {
        =
        =
        j.setJarByClass (EmpDnoSexSalSum.class);
        j.setMapperClass (EmpDnoSexSal.class);
        j.setReducerClass (RedForSum.class);
        =
        =
    }
}
```

Driver class	Mapper class	Reducer class	hql statement
Driver11.class	EmpDnoSexSal.class	RedForSum.class	select dno, sex, sum(sal) from emp group by dno, sex
Driver12.class	EmpDnoSexSal.class	RedForAvg.class	select dno, sex, avg(sal) from emp group by dno, sex
Driver13.class	EmpDnoSexSal.class	RedForMax.class	select dno, sex, max(sal) from emp group by dno, sex;
Driver14.class	EmpDnoSexSal.class	RedForMin.class	select dno, sex, min(sal) from emp group by dno, sex;
Driver15.class	EmpDnoSexSal.class	RedForCnt.class	select dno, sex, count(*) from emp group by dno, sex;

Up to now, we did single grouping and single aggregations, multi grouping and single aggregations.

How to perform multiple Aggregations?

ex:-

```
hive> select sex, sum(sal), avg(sal),
        max(sal), min(sal), count(*)
          from emp
            group by sex;
```

→ In this statement number of grouping columns 1.
number of aggregations are 5 (multiple).

→ First find,
Sum, avg, max, min, count all aggregations in one Reducer,
before writing Concatinate all aggregated values as single value.

ex:-

```
String res = sum + "\t" + avg + "\t" +
             max + "\t" + min + "\t" + cnt;
```

Write res as value.

We will write Reducer for All aggregations

RedForAll.java.

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class RedForAll extends Reducer<
    Text, IntWritable, Text, Text>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
      Context con) throws IOException, InterruptedException
    {
        int tot=0;
        int avg max=0;
        int min=0;
        int cnt=0;
      for(IntWritable v:vlist)
      {   int val=v.get();
          cnt++;
        if (cnt==1) { max=val; min=val; }
          max = Math.max(max, val);
          min = Math.min(min, val);
          tot += val;
      }
      int avg = tot/cnt;
```

→

MR-X16

```
String res = tot + "\t" +
             avg + "\t" +
             max + "\t" +
             min + "\t" +
             cnt;

con.write(k, new Text(res));
} // end of reduce()
} // end of RedForAll
```

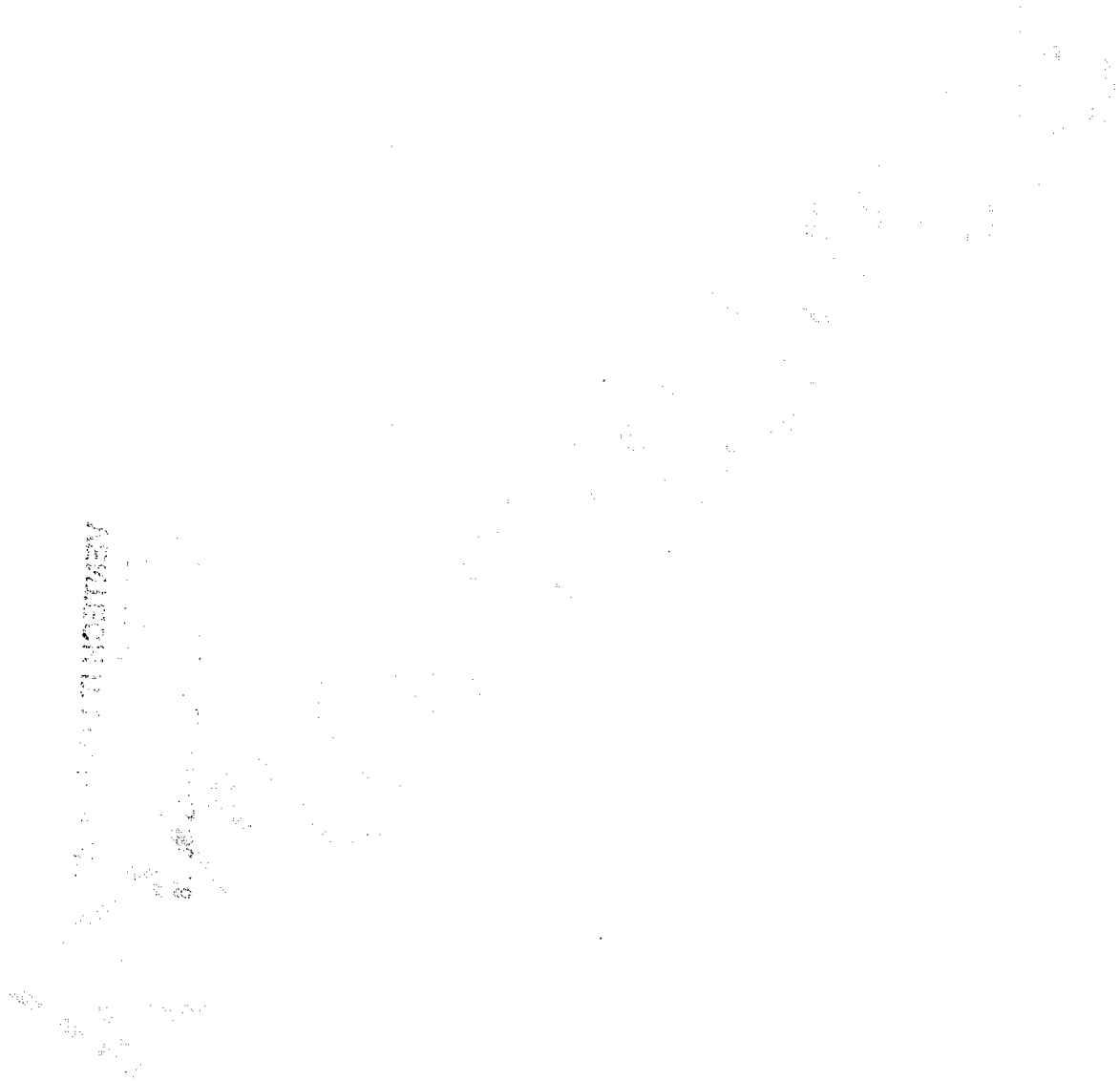
Driver16.java

Task:-
for each sex group all aggregation on salary is required)

```
package mr.analytics;
=
public class Driver16
{
    public static void main(String[] args) throws Exception
    {
        =
        j.setJarByClass(Driver16.class);
        j.setMapperClass(EmpSexSal.class);
        j.setReducerClass(RedForAll.class);
        =
    }
}
```

SINGLE GROUPING WITH MULTIPLE AGGREGATIONS

Driver class	Mapper class	Reducer class.	HQL statement
Driver16.class	EmpSexSal.class	RedForAll.class	select sex, sum(sal), avg(sal), max(sal), min(sal), count(*) from emp group by sex;
Driver17.class	EmpDnoSal.class	RedForAll.class	select dno, sum(sal), avg(sal), max(sal), min(sal), count(*) from emp group by dno;
	MULTIPLE	GROUPING WITH	MULTIPLE AGGREGATIONS
Driver18.class	EmpDnoSexSal.class	RedForAll.class	select dno, sex, sum(sal), avg(sal), max(sal), min(sal), count(*) from emp group by dno, sex;



In All previous examples,
we worked with grouping aggregations.
then how to perform entire column aggregations.

eg:-

```
select sum(sal) from emp;
select sum(price*amt) from sales;
select avg(sal) from emp;
select max(sal) from emp;
select min(sal) from emp;
select count(*) from emp;
```

→ In all above queries,
we need one aggregated value
for entire column.

Example:

I/p file → /user/training/mrdemo/emp

Schema → ecode, name, sal, sex, dno

~~In Mapper~~

Task → Total Salary of entire file.

In Mapper,

provide a Common key for each row.
and write Salary as value.

ex:-

I/p file:-

emp

101, Amar, 20000, M, 11
102, Amala, 30000, F, 12
103, Ankit, 40000, M, 12
104, Ankita, 50000, F, 11
105, Anuz, 70000, M, 11

Ventech	20000
Ventech	30000
Ventech	40000
Ventech	50000
Ventech	70000

Key is same for each row.

When this o/p is sent to Reducer All salaries formed as an group. Because key is same (one group)

EmpSal.java

```
package mr.analytics;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import java.io.IOException;
public class EmpSal extends Mapper<LongWritable,
        Text, Text, IntWritable>
{
    public void map(LongWritable k, Text v, Context con)
        throws IOException, InterruptedException
    {
        String line = v.toString();
        String[] w = line.split(",");
        int sal = Integer.parseInt(w[2]);
        con.write(new Text("ventech"),
                  new IntWritable(sal));
    }
}
```

Here, in this example "ventech" is key for each row.

Already we have reducer class for SUM. [RedForSum]

```
Package mr.analytics;
  ≡
Public class AggrSum extends .
{
  public static void main(String[] args) throws Exception
  {
    j.setJarByClass(AggrSum.class);
    j.setMapperClass(AggrA.EmpSal.class);
    j.setReducerClass(RedForSum.class);
      =
  }
}
```

When you submit the driver (above) the o/p will be

o/p

ventech 210000

→ o/p key → total salary.

→ But I dont want o/p key

only output should be

210000

then you have to write NullWritable as key in you reducer.

→

(see next page for Reducer change)

RedSum.java

```
package mr.analytics;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import java.io.IOException;
public class RedSum extends Reducer<Text, IntWritable,
                    NullWritable, IntWritable>
{
    public void reduce(Text k, Iterable<IntWritable> vlist,
    Context con) throws IOException, InterruptedException
    {
        int tot=0;
        for (IntWritable v: vlist)
            tot += v.get();
        con.write(NullWritable.get(), new IntWritable(tot));
    }
}
```

Same style create Reducers for AVG/MAX/MIN/MAX/CNT

ex:-

RedAvg.class
RedMax.class
RedMin.class
RedCnt.class
RedAll.class → for All Aggregations.

Driver class	Mapper class	Reducer class	hql statement
Driver19.class	EmpSal.class	RedSum.class	select sum(sal) from Emp;
Driver20.class	EmpSal.class	RedAvg.class	select avg(sal) from Emp;
Driver21.class	EmpSal.class	RedMax.class	select max(sal) from Emp;
Driver22.class	EmpSal.class	RedMin.class	select min(sal) from Emp;
Driver23.class	EmpSal.class	~~RedAll.class~~ RedCnt.class	select count(*) from Emp;
Driver24.class	EmpSal.class	RedAll.class	select sum(sal), avg(sal), max(sal), min(sal), ~~avg~~ count(*) from emp;

Filtering rows.

Task :- from employee file, need to write only Female data into output file.

hql :- hive> select * from emp where sex='F';

RowFilter1.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class RowFilter1 extends Mapper<LongWritable,
        Text, Text, NullWritable>
{       //emp -> ecode, ename, esal, sex, dno
    public void map(LongWritable k, Text v,
                    Context con) throws IOException,
                                        InterruptedException
    {
        String[] w = v.toString().split(",");
        String sex = w[3]; ---> is a regular expression,
        if(sex.matches("[F,f]")      ("F" or "f"
            con.write(v, NullWritable.get());
    }
}
```

In job definition, (In Driver class),
we need to suspend the reducer,
otherwise default reducer will be activated.

Driver25.java.

```
package mr.analytics;
=
=
public class Driver25
{
    public static void main(String[] args) throws Exception
                                                   Exception
    {
        =
        =j.setJarByClass(Driver25.class);
        j.setMapperClass(RowFilter1.class);
        j.setNumReduceTasks(0); // suspending Reducer
        =
        =
    }
}
```

MR - Y2

Inputfile :- ~~Text1.txt~~

```
$ cat > comment.txt
hadoop is big data framework
does hadoop fits for oLTP
HANA can be used for realtime aswell as batch
But Hadoop is a great speed System for Batch process.
Upto certain limit of data HANA is good not for petabytes.
```

Task:-

In above file, there are 5 lines.

1, 2, 4 lines are about hadoop

remaining lines are about others.

Now, we need to write only hadoop lines in o/p file.

RowFilter2.java

```
package mr.analytics;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import java.io.IOException;
import java.util.StringTokenizer;
public class RowFilter2 extends Mapper<LongWritable,
        Text, Text, NullWritable>
{
```

→

```
public void map(LongWritable k, Text v,
    Context con) throws IOException, InterruptedException
{
    String line = v.toString();
    StringTokenizer t = new StringTokenizer(line);
    int cnt = 0;
    while (t.hasMoreTokens())
    {
        String word = t.nextToken();
        if (word.matches("[H,h]adoop"))
            cnt++;
    }
    /*
       if line has Hadoop/hadoop word
          cnt value will be greater than 0.
    */
    if (cnt > 0)
        con.write(v, NullWritable.get());
  }
}
```

Driver26.Java

```
public class Driver26
{
    public static void main(String[] args) throws Exception
    {
        Job j = new Job();
        j.setJarByClass(Driver26.class);
        j.setMapperClass(RowFilter2.class);
        j.setNumReduceTasks(0);
    }
}
```

MR-23

ColumnFilter

Input file: probile.txt

schema → id, name, email, phone, city, designation, company, experience (8 fields)

delimiter → "," [comma]

Task → Write only name, email, experience into o/p file.

hql

```
hive> select name, email, experience
      from probile;
```

ColumnFilter.java


```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import java.io.IOException;
public class ColumnFilter extends Mapper <LongWritable,
    Text, Text, NullWritable>
{
```

→

```
public void map(LongWritable k,
   Text v, Context con) throws IOException,
InterruptedException
{
    String line = v.toString();
    String[] words = line.split(",");
    string name = words[1];
    String email = words[2];
    String experience = words[7];
    String newLine = name+","+email+","+
                     experience;
    /* Sample row
         Ravi, Ravi@gmail.com, 5
    */
    con.write(new Text(newLine), NullWritable.get());
  }
}
```

Driver27.java

```
public class Driver27
{
   public static void main(String[] args) throws
                                        Exception
   {
      j.setJarByClass(Driver27.class);
      j.setMapperClass(ColumnFilter.class);
      j.setNumReduceTasks(0);
   }
}
```

Generating New Fields

Input File: emp.txt

Schema → ecode, name, sal, ~~dno~~ sex, dno

Sample data → 101, Amar, 20000, M, 11
102, Amala, 30000, F, 12
⋮

Task :- foreach row, tax @ 10%, hra @ 20%. and netsalary to be generated.

hql :-

```
hive> select ecode, name, sal,
      sal*0.1 as tax,
      sal*0.2 as hra,
      sal+(sal*0.2)-(sal*0.1) as net,
      sex, dno from emp;
```

GenerateColumns.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
import ~~org~~ java.io.IOException;
```

```
public class GenerateColumns extends Mapper<
    LongWritable, Text, Text, NullWritable>
{
    public void map(LongWritable k, Text v,
        Context con) throws IOException, InterruptedException
    {
        String line = v.toString();
        String[] words = line.split();
        int sal = Integer.parseInt(words[2]);
        int tax = (sal * 10/100);
        int hra = sal * 20/100;
        int net = sal + hra - tax;
        String newLine = W[0] + "," +
                         W[1] + "," +
                         sal + "," +
                         tax + "," +
                         hra + "," +
                         net + "," +
                         sex + "," +
                         dno;
        /*
          Sample row:
          101, Amar, 20000, 2000, 4000, 22000, M, 11
        */
        con.write(new Text(newLine), NullWritable.get());
    }
}
```

Driver28.java

```
=
=
public class Driver28
{
    public static void main (String[] args)
        throws Exception
    {
        =
        =
        j.setJarByClass (Driver28.class);
        j.setMapperClass (GenerateColumns.class);
        j.setNumReduceTasks (0);
        =
        =
    }
}
```


In all previous examples, in which we suspended reducer,

the following settings are not required in Driver class.

```
j.setOutputKeyClass (———);
j.setOutputValueClass (———);
```

→ required only when Reducer is involving.

Transformations

Input file :— emp.txt

Schema :— ecode, name, sal, sex, dno

Task :

(i) foreach row, based on salary, Grade to be generated.

ex:—

sal >= 70000 ⟶ "A"

sal >= 50000 and sal < 70000 ⟶ "B"

sal >= 30000 and sal < 50000 ⟶ "C"

sal < 30000 ⟶ "D"

(ii) **Sex**

"M" —as→ "Male"

"F" —as→ "Female"

(iii) dno's to be transformed as department-names.

ex:—

11 ⟶ "Marketing"

12 ⟶ "HR"

13 ⟶ "Finance"

14, 15, ... ⟶ "other"

MR-YG.

hql:-

```
hive> select ecode, ename, sal,
      if(sal>=70000,'A',
         if(sal>=50000,'B',
            if(sal>=30000,'C','D'))) as grade,
      if(sex='F','Female','Male') as sex,
      if(dno=11,'Marketing',
         if(dno=12,'HR',
            if(dno=13,'Finance','other')))
      as dname from emp;
```


Sample Input row:-

101, Amar, 20000, M, 11

o/p expected:

101, Amar, 20000, D, Male, Marketing

Transform.java

```
package mr.analytics;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
```

```
public class Transform extends Mapper<LongWritable,
Text, Text, NullWritable>
{
    public void map(LongWritable k,
      Text v, Context con) throws IOException,
                        InterruptedException
    {
        String line = v.toString();
      String[] w = line.split(",");
      int sal = Integer.parseInt(w[2]);
      int
      String sex = w[3];
      int dno = Integer.parseInt(w[4]);
      String grade;
      if (sal >= 70000)
            grade = "A";
      else if (sal >= 50000)
            grade = "B";
          else if(sal >= 30000)
              grade = "C";
            else
              grade = "D";
      if (sex.matches("[F,f]"))
          sex = "Female";
      else
          Sex = "Male";
```


→

```
String dname;
switch (dno)
{
   case 11 :
         dname = "Marketing";
         break;
   case 12:
         dname = "HR";
         break;

   case 13:
         dname = "Finance";
         break;
   default:
         dname = "other";
}
String newLine = W[0] + "," +
                 W[1] + "," +
                 sal + "," +
                 grade + "," +
                 sex + "," +
                 dname;
con.write(new Text(newLine), NullWritable.get());
}
}
```

Driver29.java

```
public class Driver29
{
    public static void main(String[] args)
            throws Exception
    {
        =
        =
        j.setJarByClass(Driver29.class);
        j.setMapperClass(Transform.class);
        j.setNumReduceTasks(0);
        =
        =
    }
}
```

Working with Multiple Input Files.

In All previous examples we worked with single input file.

what if, if your input data is multiple input files. that too if each input file has different schema.

Example

Input file1 → emp1.txt

Schema → ecode, name, sal, ~~dn~~ sex, dno
delimiter → Comma (,)

Input file2 → emp2.txt

Schema → ecode, name, dno, sex, sal
Delimiter → Comma (,)

Input file3 → emp3.txt

Schema → ecode, name, sal, sex, dno
Delimiter → Comma (,)

Now we need to use two mappers

Mapper1 is for emp1.txt and emp3.txt because schema is same.

Mapper2 is for emp2.txt, different schema In Mapper2, we need reformat the schema of file (emp2.txt) same as emp1.txt and emp3.txt.

Task :-

need to merge all rows of emp1.txt, emp2.txt and emp3.txt into a single o/p file.

hql :

```
hive> select * from (
        select ecode,name,sal,sex,dno from emp1
          union all
        select ecode, name, sal, sex, dno from emp2
          union all
        select ecode, name, sal, sal, dno from emp3) e;
```

MergeMap1.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
import ~~org.apache.hadoop~~ java.io.IOException;
public class MergeMap1 extends Mapper<LongWritable,
  Text, Text, NullWritable>
{
```

→

```
public void map(LongWritable k, Text v,
  Context con) throws IOException, InterruptedException
{
    /*
      We need not to change schema of
      emp1.txt and emp3.txt.
      So, as it is, we write same record
      as key and NullWritable as value
    */
     con.write(v, NullWritable.get());
}
}
```


MergeMap2.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class MergeMap2 extends Mapper<LongWritable,
    Text, Text, NullWritable>
{
```

→

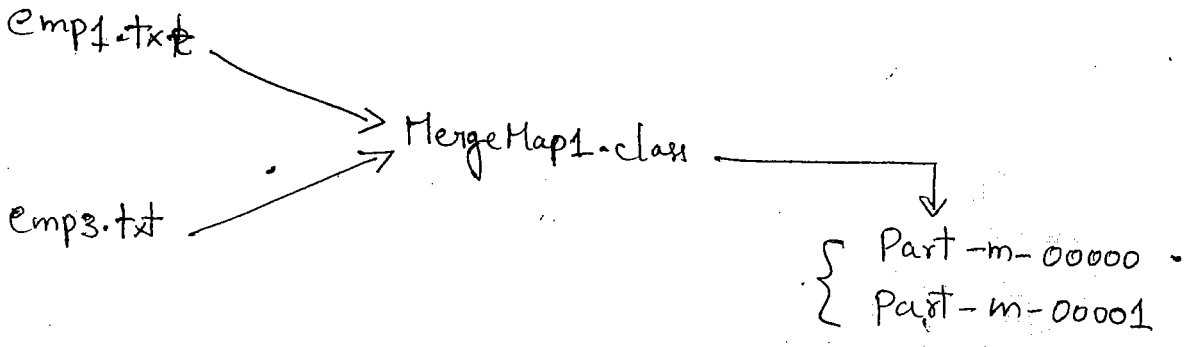
```
public void map(LongWritable k, Text v,
    Context con) throws IOException, InterruptedException
{
    String line = v.toString();
    String[] w = line.split(",");
    /*
       emp2.txt schema is ecode, name, dno, sex, sal
       we need to convert it as
                    ecode, name, sal, sex, dno
    */
    String newLine = w[0] + "," +
                     w[1] + "," +
                     w[4] + "," +
                     w[3] + "," +
                     w[2];
    con.write(new Text(newLine), NullWritable.get());
}
}
```

now we have two Mappers

MergeMap1 ⟶ fd emp1.txt and emp3.txt

& MergeMap2 ⟶ fd emp2.txt

But if you Suspend Reducer



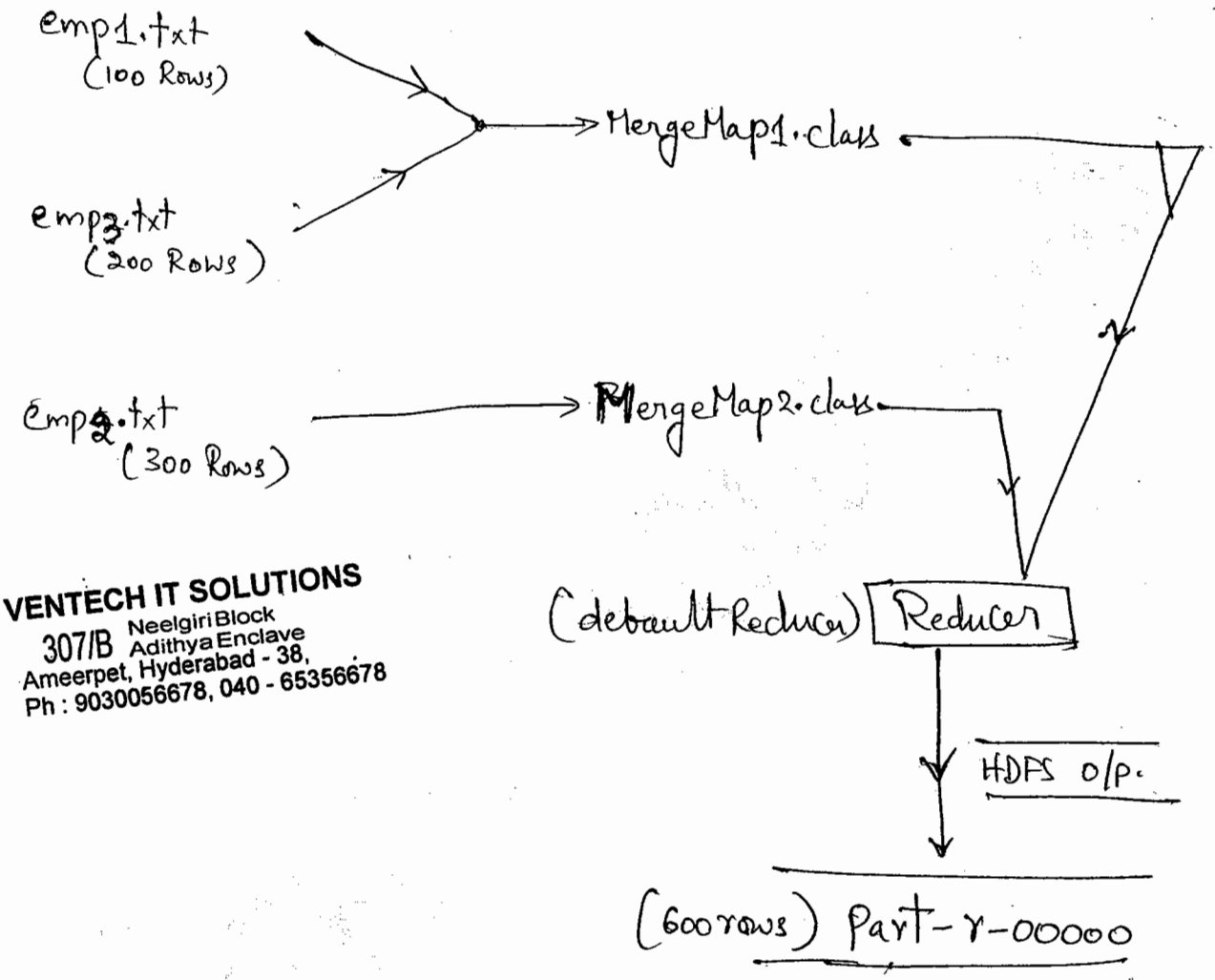
emp2.txt ⟶ MergeMap2.class ⟶ part-m-00002

that means, MergeMap1 executed for two times,
MergeMap2 executed for one time.

So, 3 seperate o/p files we got.
But our task is to merge all rows into one o/p file.
So we need Reducer. But in Reducer, we dont want any functionality. So If you dont use any reducer, (But dont suspend Reducer) default Reducer will Come in to picture.

This simply collects o/p's of all Mappers and writes into HDFS file (single)

⟶



Then in Driver class, how to specify multiple input files and multiple Mappers?

- how to assign each Input file to a seperate Mapper.

→

Driver ⟶ Driver30.java

```
Package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
public class Driver30
{
    public static void main(String[] args) throws Exception
    {
        Configuration c = new Configuration();
        Job j = new Job(c, "Merge");
        j.setJarByClass(Driver30.class);
        j.set
        j.setOutputKeyClass(Text.class);
        j.setOutputValueClass(NullWritable.class);
        Path p1 = new Path(args[0]); //emp1.txt
        Path p2 = new Path(args[1]); //emp2.txt
        Path p3 = new Path(args[2]); //emp3.txt
        Path p4 = new Path(args[3]); //output directory
```

```
MultipleInputs.addInputPath (j, P1,
                TextInputFormat.class, MergeMap1.class);
MultipleInputs.addInputPath (j, P2, TextInputFormat.class,
                MergeMap2.class);
MultipleInputs.addInputPath (j, P3, TextInputFormat.class,
                MergeMap1.class);

/*
   In above statements
      we assigned P1, P3 to MergeMap1.class
      and P2 to MergeMap2.class
   then two Mappers will write to default Reducer
   because we did not set any reducer
*/

FileOutputFormat.setOutputPath (j, P4);
System.exit (j.waitForCompletion (true) ? 0:1);
} // end of main
} // end of Driver30
```

Now you have two Mappers

MergeMap1.class

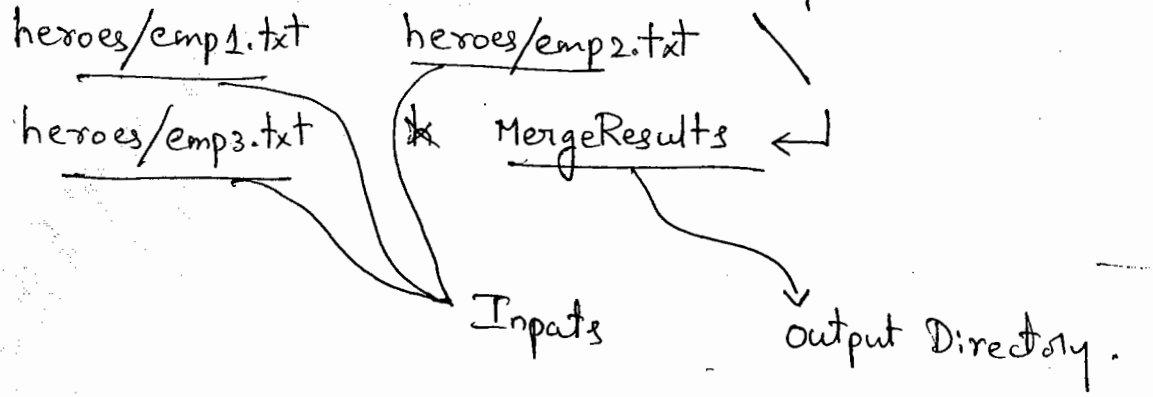
MergeMap2.class

and Driver class, Driver30.class

export these classes into /home/training/Desktop/mrdemo.jar

Submit Job.

```
$ hadoop fs -mkdir heroes ↵
$ hadoop fs -copyFromLocal emp1.txt heroes ↵
$ hadoop fs -copyFromLocal emp2.txt heroes ↵
$ hadoop fs -copyFromLocal emp3.txt heroes ↵
$ hadoop jar Desktop/mrdemo.jar mr.analytics.Driver30 \
```



Check the results

```
$ hadoop fs -cat MergeResults/part-r-00000 ↵
```

Merged Aggregations.

Already you have 3 input files emp1.txt, emp2.txt, emp3.txt. (schemas already described in previous example)
Think, each file is imported from 3 different branch (OLTP) Servers.

TASK: From All 3 branches. What is total salary for each sex group.

hql:-

```
hive> select sex, sum(sal) from
        (
          select sex, sal from emp1
                union all
          select sex, sal from emp2
                union all
          select sex, sal from emp3) e
          group by sex;
```

Now we need two seperate Mappers

SexSalMap1 → for emp1.txt and emp3.txt

SexSalMap2 → for emp2.txt

to perform Sum Aggregation, we need Reducer class already we have RedForSum.class

SexSalMap1.java

```
package mr.analytics;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class SexSalMap1 extends Mapper<LongWritable,
    Text, Text, IntWritable>
{       // ecode, name, sal, sex, dno
    public void map(LongWritable k, Text v,
        Context con) throws IOException, InterruptedException
    {
        String line = v.toString();
        String[] w = line.split(",");
        String sex = w[3];
        int sal = Integer.parseInt(w[2]);
        con.write(new Text(sex), new IntWritable(sal));
    }
}
```

SexSalMap2.Java

```
package mr.analytics;
=
=
=
=   } imports

public class SexSalMap2 extends Mapper <LongWritable,
    Text, Text, IntWritable>
{   // emp2.txt → ecode, name, dno, sex, sal
    public void map(LongWritable k, Text v,
        Context con) throws IOException, InterruptedExce-
                                              -ption
    {
        String line = v.toString();
        String[] w = line.split(",");
        String sex = w[3];
        int sal = Integer.parseInt(w[4]);
        con.write(new Text(sex), new IntWritable(
                                          sal));
    }
}
```

→

Driver ⟶ Driver31.java

```
public class Driver31
{
    public static void main(String[] args)
            throws Exception
    {
        Job j = new Job();
        j.setJarByClass(Driver31.class);
        j.setReducerClass(RedForSum.class);
        j.setOutputKeyClass(Text.class);
        j.setOutputValueClass(IntWritable.class);
        Path p1 = new Path(args[0]); //emp1.txt
        Path p2 = new Path(args[1]); //emp2.txt
        Path p3 = new Path(args[2]); //emp3.txt
        Path p4 = new Path(args[3]); //output directory
    MultipleInputs.addInputPath(j, P1, TextInputFormat.class,
                                SexSalMap1.class);
    MultipleInputs.addInputPath(j, P2, TextInputFormat.class,
                                SexSalMap2.class);
    MultipleInputs.addInputPath(j, P3, TextInputFormat.class,
                                SexSalMap1.class);
    FileOutputFormat.setOutputPath(j, P4);
    }
}
```

Branch (File) Aggregations

TASK

foreach Branch what is total Salary.

hql :-

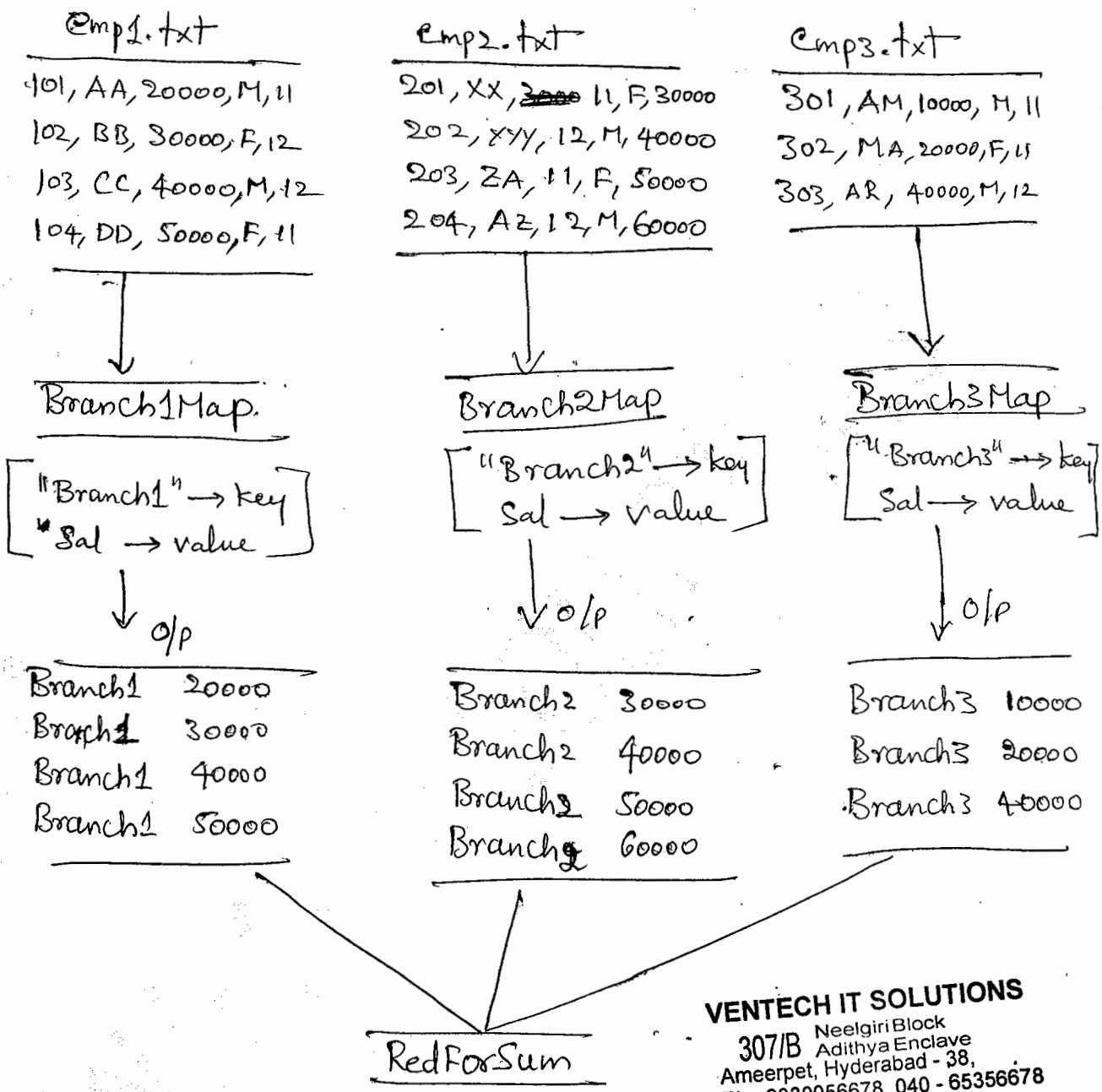
```
hive> select branch, Sum(sal) from
      (
        select 'branch1' as branch, sal from emp1
            union all
        select 'branch2' as branch, sal from emp2
            union all
        select 'branch3' as branch, sal from emp3
      ) e group by branch;
```

We need 3 seperate Mappers.

emp1 → Branch1Map → "Branch1" as key
Sal as value

emp2 → Branch2Map → "Branch2" as key
Sal as value

emp3 → Branch3Map → "Branch3" as key
Sal as value.



(i) grouping

k	vlist
Branch1	<20000, 30000, 40000, 50000>
Branch2	<30000, 40000, 50000, 60000>
Branch3	<10000, 20000, 40000>

(ii) sort

=

(iii) sum (Aggregation)

↓ HDFS o/p

Branch1	140000
Branch2	180000
Branch3	70000

Branch1Map.java

```
package .mra analytics;

public class Branch1Map extends Mapper <LongWritable,
  Text, Text, IntWritable>
{ //emp1 → ecode, ename, sal, sex, dno
  public void map(LongWritable k, Text v, Context con)
    throws IOException, InterruptedExceptim
    {
      String line = v.toString();
      String[] w = line.split(",");
      int sal = Integer.parseInt(w[2]);
      con.write(new Text("Branch1"), new IntWritable(sal));
    }
}
```

Branch2Map.java

```
package mr.analytics;

public class Branch2Map extends Mapper < LongWritable,
  Text, Text, IntWritable>
{
  //emp2 → ecode, ename, ~~sal~~ dno, sex, sal
  public void map(LongWritable k, Text v, Context con)
    throws IOException, InterruptedException
    {
      String line = v.toString();
      String[] w = line.split(",");
      int sal = Integer.parseInt(w[4]);
      con.write(new Text("Branch2"), new IntWritable(sal));
    }
}
```

Branch3Map.java

```
package mr.analytics;

public class Branch3Map extends Mapper<LongWritable,
  Text, Text, IntWritable>
 { // emp3.txt → ecode, name, sal, sex, dno
    public void map(LongWritable k, Text v,
      Context con) throws IOException, InterruptedException
     {
       String line = v.toString();
       String[] w = line.split(",");
       int sal = Integer.parseInt(w[2]);
       con.write(new Text("Branch3"), new IntWritable(sal));
     }
 }
```

Driver32.java

```
package mr.analytics;

public class Driver32
 {
    public static void main(String[] args) throws Exception
     {
        j.setReducerClass(RedForSum.class);

  MultipleInputs.addInputPath(j, p1, TextInputFormat.class, Branch1Map.class);
MultipleInputs.addInputPath(j, p2, TextInputFormat.class, Branch2Map.class);
MultipleInputs.addOutputPath
MultipleInputs.addInputPath(j, p3, TextInputFormat.class, Branch3Map.class);
  FileOutputFormat.addOutputPath(j, p4);
 }
}
```

TASK:

Eliminating Duplicate rows based on entire row match

Input file : Samp.txt

Schema : ecode, name, sal, sex, dno

data :

101, AA, 20000, M, 11
102, BB, 30000, F, 12
101, AA, 20000, M, 11
103, CC, 40000, M, 11
102, BB, 30000, F, 12
101, AA, 20000, M, 11

In above file, 101 record repeated for 3 times.
102 " " 2 times.
103 " " only for 1 time.

O/p expected:

101, AA, 20000, M, 11
102, BB, 30000, F, 12
103, CC, 40000, M, 11

Mapper logic :-

Write the row(done) as key and NullWritable as value.

Mapper O/p.

k	v
101, AA, 20000, M, 11	NullWritable.get()
102, BB, 30000, F, 12	"
101, AA, 20000, M, 11	"
103, CC, 40000, M, 11	"
102, BB, 30000, F, 12	"
101, AA, 20000, M, 11	"

When this Mapper o/p is sent to Reducer,

(i) grouping

K	vlist → NullWritable
101, AA, 20000, M, 11	<NW, NW, NW>
102, BB, 30000, F, 12	<NW, NW>
103, CC, 40000, M, 11	<NW>

(ii) Sort.

(iii) In reduce() function,

```
public void reduce(Text k, Iterable<NullWritable> vlist,
                   Context con) throws - ---
{
    con.write(k, NullWritable.get());
}
```

In grouping phase, you have 3 groups. Because 3 unique records.

So, reduce() function executed for 3 times. at each iteration, we write ~~key~~ Record as key and NullWritable as value.

So o/p ↓

101, AA, 20000, M, 11
102, BB, 30000, F, 12
103, CC, 40000, M, 11

NODupeRowMap.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class NoDupeRowMap extends Mapper<LongWritable,
    Text, Text, NullWritable>
{
    public void map(LongWritable k, Text v,
      Context con) throws IOException, InterruptedException
    {
        con.write(v, NullWritable.get());
    }
}
```

RedForNoDupeRow.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class RedForNoDupeRow extends Reducer<Text,
  NullWritable, Text, NullWritable>
{
```

```
    public void reduce(Text k, Iterable<NullWritable> vlist,
            Context con) throws IOException, InterruptedException
    {
        con.write(k, NullWritable.get());
    }
}
```

Driver → Driver33.java

```
package mr.analytics;
=
public class Driver33
{
    public static void main(String[] args)
      throws IOException, InterruptedException Exception
    {
        =
        j.setJarByClass(Driver33.class);
        j.setMapperClass(NoDupeRowMap.class);
        j.setReducerClass(RedForNoDupeRow.class);
        =
    }
}
```

Previous example, will eliminate duplicates based on entire row match.

But I want to eliminate duplicate rows based on key match.

Input file ⟶ file1.txt

Schema ⟶ prid, price

data ⟶

P1, 1000 ✓
P2, 2000 ✓
P3, 5000 ✓
} first Duplicates.
P1, 1200 ✗
P2, 2500 ✗
P2, 3000 ✗
P1, 1100 ✗
P1, 900 ✗

from each Duplicate list, firstDuplicate is needed.

o/p Expected.

P1, 1000
P2, 2000
P3, 5000

Mapper logic:

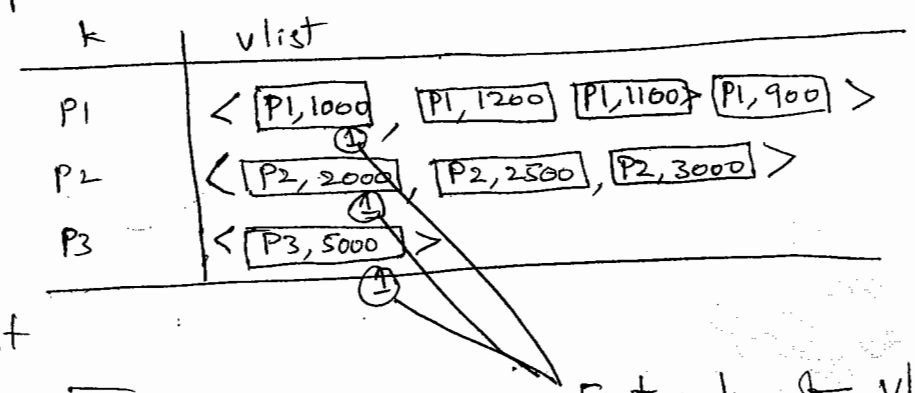
Write prid as key and record(line) as value.

↓o/p.

k	v
P1	P1, 1000
P2	P2, 2000
P3	P3, 5000
P1	P1, 1200
P2	P2, 2500
P2	P2, 3000
P1	P1, 1100
P1	P1, 900

When this Mapper o/p is sent to Reducer,

(i) group



(ii) Sort

First value of vlist

(iii) reduce()

In reduce() function, from vlist write only first value as key, NullWritable as value.

o/p →

P1,1000
P2,2000
P3,5000

NoDupekeyMapper.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class NoDupekeyMapper extends Mapper<LongWritable,
    Text, Text, Text>
{
```

→

```
public void map(LongWritable k, Text v, Context con)
   throws IOException, InterruptedException
{ // file1.txt ⟶ prid, price
      String line = v.toString();
      String[] w = line.split(",");
      String prid = w[0];                    ⟶ entire line writing as value.
      con.write(new Text(prid), v);
 }
}
```

RedForFirstDuplicate.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class RedForFirstDuplicate extends Reducer<Text, Text,
      Text, NullWritable>
{
   public void reduce(Text k, Text Iterable<Text> vlist,
      Context con) throws IOException, InterruptedException
   {
         for(Text v:vlist)
         {
            con.write(v, NullWritable.get());
            break;
         }
   }
}
```

if you use

Driver → to get eliminate duplicate rows based on key. [to get first Duplicate]

Driver → Driver34.java

```
package mr.analytics;

import ...

public class Driver34
{
    public static void main(String[] args)
        throws Exception
    {
        ...
        j.setJarByClass(Driver34.class);
        j.setMapperClass(NoDupeKeyMapper.class);
        j.setReducerClass(RedForFirstDuplicate.class);

        ...
    }
}
```

Then, How to get last duplicate row, based on key.

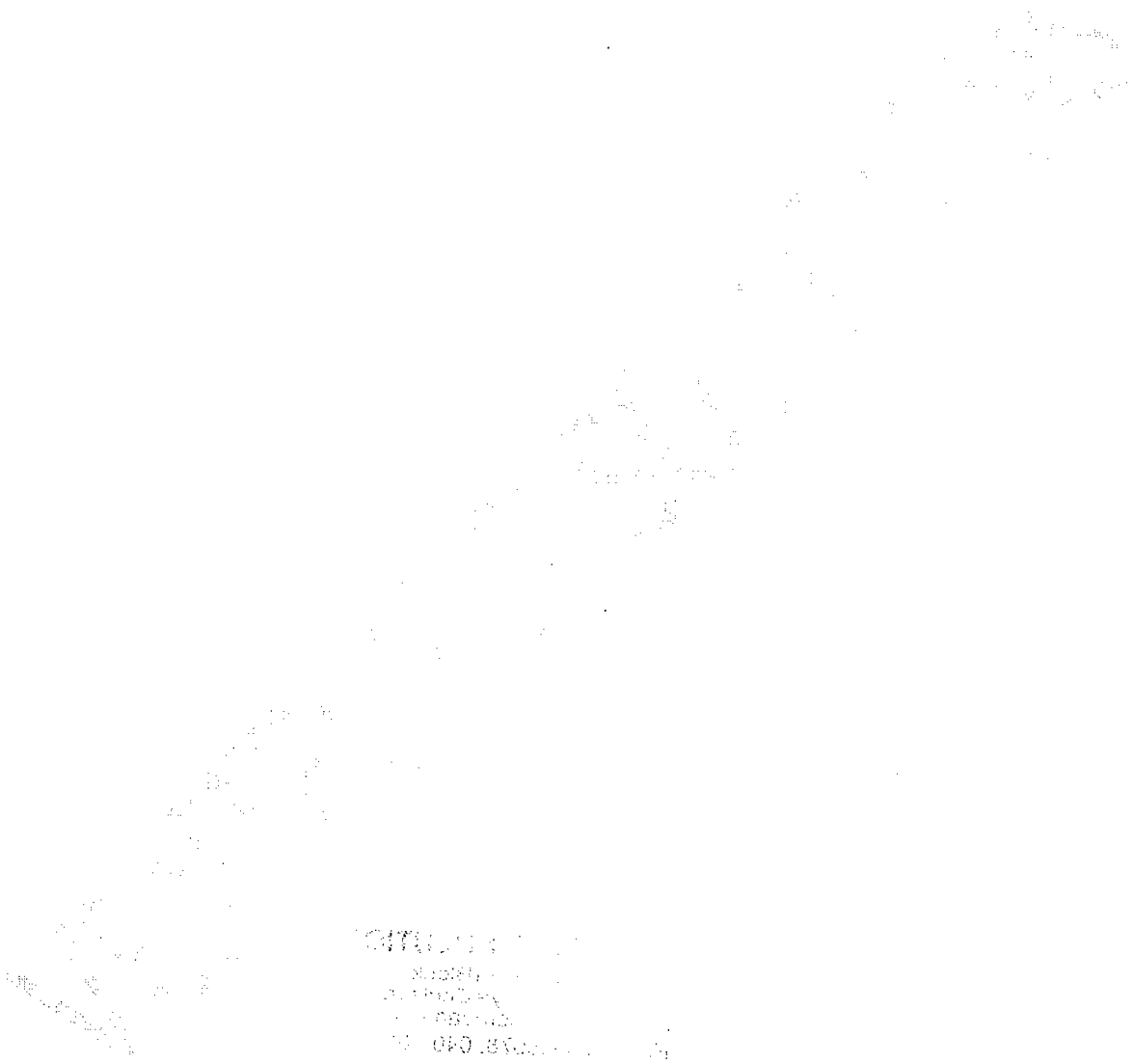
RedForLastDuplicate.java

```
package mr.analytics;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
public class RedForLastDuplicate extends Reducer<Text,
    Text, Text, NullWritable>
{
    public void reduce(Text v, Iterable<Text> vlist,
        Context con) throws IOException, Interrupted-
                                        -Exception
    {
        String rec="";
        for(Text v: vlist)
        {
            rec = v.toString()
        }
        con.write(new Text(rec), NullWritable.get());
    }
}
```

Driver → to eliminate duplicate rows based on key [to get Last Duplicate]

Driver → Driver35.java

```
package mr.analytics;
=
=
public class Driver35
{
    public static void main(String[] args)
        throws Exception
    {
        =
        j.setJarByClass(Driver35.class);
        j.setMapperClass(NoDupeKeyMapper.class);
        j.setReducerClass(RedForLastDuplicate.class);
        =
        =
    }
}
```



Writables in Hadoop

up to now, we have used different HDFS data types such as LongWritable, IntWritable, Text, FloatWritable, NullWritable - - - etc.

In this way, hadoop supports equivalant hdfs data types for each java primitive types. (int, long, float, String etc)

All these are called Writables in hadoop.

What is a Writable in hadoop?

→ A Writable is an interface in hadoop.

→ hdfs types must implement this interface.

→ Hadoop provides these Writable wrappers for almost all java primitives types and some other types.

Why hadoop is using Writables?

We know, In hadoop data is distributed as blocks into different slave nodes.
While reading & writing data into hadoop cluster, data needs to be transmitted between different nodes in a distributed environment.

This process requires, Serialization and deserialization of data to convert the data that is in structured format to bytestream and byte stream to structured format.

Therefore hadoop uses an efficient serialization protocol to serialize data between mapper and Reducer phases. These Serialization protocols are called Writables.

ex:- IntWritables, LongWritable, Text, BooleanWritable etc.

In WordCount example & SexSal Aggregations example, write write Text as key, and IntWritable as value, in Mapper Class.

WordCount → con.write(new Text(word), new IntWritable(1));

(select sex, sum(sal) from emp group by sex;
← EmpSexSalSum → con.write(new Text(sex), new IntWritable(1));

and from Reducers, we write Text as key, Sum value as Value

WordCount → con.write(~~new Text(word),~~ k, new IntWritable(tot));
k → word

EmpSexSalSum → con.write(~~new Text(s~~ k, new IntWritable(tot));
k → Sex

In this way, hadoop supports many primitive Writables. But these can be used for simple Applications of operations.

Consider the following scenario

I would like to transmit nativeCity, workingCity, Country combination as key in Mappers/Reducers.

ofcours, you can concatinate them as string, and you can make the String as key.

ex:-

```
String into = nativeCity + "\t" + workingCity
                + "\t" + Country ;
con.write(new Text(into), new IntWritable(val));
```

when this data is sent Reducer,

if you want to manipulate the key, again you need to split it and proces.

and one more important thing is,

In Reducer, in Sort phase, Reducer uses default Sort mechanism,

what if, if I want to implement secondary Sort, such as nativeCity in assending order, workingCity in descending order and Country in assending order. For such things, primitive Writables can not say answer. so we need to write our own

Custom Writables..

Then,

How to Write, Custom Writables.

If any user defined class implements Writable interface, then the class is called Custom Writable.

Structure of Writable interface.

```
public interface Writable
{
    void readFields(DataInput in);
    void write(DataOutput out);
}
```

So any User defined class, which implementing this interface, must provide the code implementation of these two methods at least. (we need to override these two methods).

write(DataOutput out) → It is used to Serialize [object to bytestream] the fields of Object to out.

readFields(DataInput in) → It is used to deserialize [byteStream to Object] the fields of the object from in.

WritableComparable

If you want write your custom datatype as key rather that the value, then we need WritableComparable

then, User defined class has to implement WritableComparable interface.

The WritableComparable interface extends from the Writable interface and Comparable Interface.

The Structure of WritableComparable

```
public interface WritableComparable extends
                    Writable, Comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o);
}
```

compareTo(WritableComparable o)

→ It is inherited from Comparable Interface

So hadoop can sort the keys in shuffle and sort phase.

BigramCount Example.

→ By this example, you can know how to use Custom Writable.

→ foreach pair of words, Count is required.

TextPair.java

```
package mr.analytics;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
public class TextPair implements WritableComparable
                                    <TextPair>
{
 Private ~~public~~ Text first;
 private ~~public~~ Text Second;
      public TextPair (Text first, Text second)
      {
            set(first, second);
      }
     public TextPair()
      {
            set (new Text(), new Text());
      }
    public TextPair(String first, String second)
      {
            set (new Text(first), new Text(second));
      }
```

```
public Text getFirst()
{
    return first;
}
public Text getSecond()
{
    return second;
}
public void set(Text first, Text second)
{
    this.first = first;
    this.second = second;
}
public void write(DataOutput out) throws IOException
{
    first.write(out);
    second.write(out);
}
public void readFields(DataInput in) throws
                                    IOException
{
    first.readFields(in);
    second.readFields(in);
}
public String toString()
{
    return first + " " + second;
}
```

```
public in compareTo(TextPair tp)
{
    int c = first.compareTo(tp.first);
    if (c!=0)
    {
        return c;
    }
    return second.compareTo(tp.second);
}
public in hashcode()
{
    return first.hashCode()*163 + second.hashCode();
}
public boolean equals(Object o)
{
    if(o instanceof TextPair)
    {
        TextPair tp = (TextPair) o;
        return first.equals(tp.first) &&
                second.equals(tp.second);
    }
    return false;
}
}
```

Mapper → BigramMap.java

```
package mr.analytics;

public class BigramMap extends Mapper <LongWritable,
    Text, TextPair, IntWritable>
{
    private static Text lastWord = null;
    private static TextPair tp = new TextPair();
    private static Text wordText = new Text();
    private static IntWritable one = new IntWritable(1);

    public void map(LongWritable key, Text value, Context con)
        throws IOException, InterruptedException
    {
        String line = value.toString();
        line = line.replace(",", "");
        line = line.replace(".", "");
        String[] words = line.split("\\W+");
        for (String word : words))
        {
            if (lastWord == null)
            {
                lastWord = new Text(word);
            }
            else
            {
                wordText.set(word);
                tp.set(lastWord, wordText);
                con.write(tp, one);
                lastWord.set(wordText.toString());
            }
        }
    }
}
```

Reducer → BigramRed.java

```
package mr.analytics;
=
public class BigramRed extends Reducer < TextPair,
   IntWritable, Text, IntWritable >
{
    ~~private stati~~
   public void reduce (TextPair k, Iterable <IntWritable> ~~vlist~~
                       vlist, Context con)
     throws IOException, InterruptedException
   {
       int cnt = 0;
     for (IntWritable v : vlist)
           cnt += v.get();
     ~~con.write(k,~~
      con.write(new Text(k.toString()),
                          new IntWritable(cnt));
   }
}
```

Driver → Driver36.java

```
public class Driver36
{ public static void main(String[] args) throws Exception
   {
       =
       j.setJarByClass(Driver36.class);
       j.setMapperClass(BigramMap.class);
       j.setReducerClass(BigramRed.class);
       =
   }
}
```

I/P

I love u
u love him
I love all

O/P

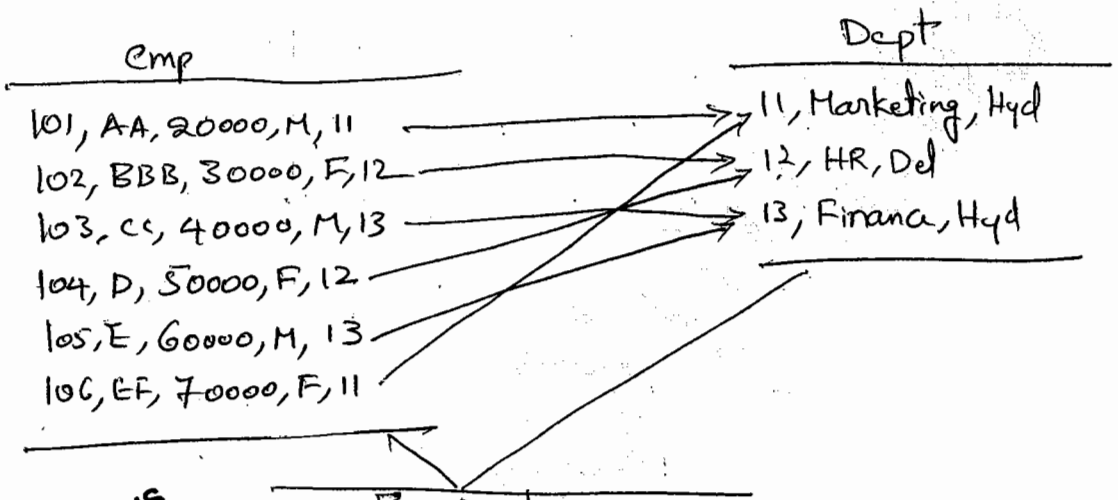
I	love	2
love	u	1
u	love	1
love	him	1

How to perform joins in MapReduce.

Input files → emp, dept

emp schema → ecode, name, sal, sex, dno

dept schema → dno, dname, dLoc



101, AA, 20000, M, 11, Marketing, Hyd
102, BBB, 30000, F, 12, HR, Del
103, CC, 40000, M, 13, Finance, Hyd
104, D, 50000, F, 12, HR, Del
105, E, 60000, M, 13, Finance, Hyd
106, EF, 70000, F, 11, Marketing, Hyd

Mapper ⟶ MapJoin.java

```
package mr.analytics;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Mapper;
public class MapJoin extends Mapper<LongWritable,
  Text, Text, NullWritable>
{
   public
    HashMap<String,String> hm = new HashMap<
                                String,String>();
    /*
       when mapper class is called,
        first setup() method executed for one time,
     and then map() " executed for
      number of input rows.
     we assign Dept file to setup(),
     and write dno as key to hashmap
     and dinfo (dname, dloc) as value
     to hashmap.
    */
```

```
public void setup(Context con) throws IOException,
                                      FileNotFoundException
{
Path[] P = DistributedCache.getLocalCacheFiles(
                        con.getConfiguration());
   /*
      we assigned Dept into DistributedCache
      in Driver class.
   */
FileInputStream fs = new FileInputStream(
                                P[0].toString());
InputStreamReader isr = new InputStreamReader(fs);
BufferedReader br = new BufferedReader(isr);
  String line;
while( (line=br.readLine())!=null)
{      //Dept → dno, dname, dloc
       String[] w = line.split(",");
        String dno = w[0];
        String dname = w[1]+","+w[2];
        hm.put(dno, dname);
}
br.close();
} // end of setup.
```

```
public void map(LongWritable k, Text v,
        Context con) throws IOException,
InterruptedException
{
    String line = v.toString();
    String[] w = line.split(",");
    String dno = w[4];
    String dinfo = hm.get(dno);
    String info = line+","+dinfo;
    con.write(new Text(info), NullWritable.get());
}
}
```


Driver ⟶ Driver37.java

```
package mr.analytics;

import org.apache.hadoop.filecache.DistributedCache;

public class Driver37
{
    public static void main(String[] args)
            throws Exception
    {
        j.setJarByClass(Driver37.class);
        j.setMapperClass(MapJoin.class);
        j.setNumReduceTasks(0);
```

emp

```
        FileInputFormat.addInputPath(j, new Path(args[0]));
    DistributedCache.addCacheFile(new Path(args[1]).toUri(),
                                  j.getConfiguration());
        FileOutputFormat.setOutputPath(j, new path(args[2]));
        System.exit(j.waitForCompletion(true) ? 0 : 1);
    }
}
```

---

export all classes into /home/training/Desktop/mrdemo.jar

Submit job as follows:

```
$ hadoop fs -ls heroes ↵
        ———— emp
        ———— dept
$ hadoop jar Desktop/mrdemo.jar \
        mr.analytics.Driver37 \
        /user/training/heroes/emp \
        /user/training/heroes/dept \
        /user/training/heroes/output ↵
```

Bharat

Pig :

---

is a data flow language in hadoop framework.

other dataflow languages: cascading, Crunch...

Cascading , Cruch are Api s.

Cascading can be done by java, python , scala.

Cascading and Scala library is called Scalding.

Crunch also same as cascading.

Pig has "Pig Latin" , is scripting language, to process data.

dataflow:- is a collection of pipes.

pipe is an operation.

operation can be loading, processing each tupie, sorting, grouping, aggregating.....

From one pipe to another pipe data is tranforming from one state to another state.

ex:

step1 --- data loading

step2 ---- devide the line into words.

step3 --- grouping the words.

step4 ---- find count aggregation for each word.

step5 -- storing the results.

Pig has two start-up modes:

i) local mode

ii) hdfs mode

i)local mode:

[training@localhost ~]$ pig -x local

grunt> lines = load 'comment' as (line:chararray);

grunt> dump lines;

[training@localhost ~]$ ls locals1

part-m-00000 _SUCCESS

[training@localhost ~]$ cat locals1/part-m-00000

i love india

i love hadoop

hadoop does not love

-- that means, if pig started local mode, it takes input from local file and writes output into local.

-- this mode is helpful in testing.

ii)Hdfs mode:

[training@localhost ~]$ pig

grunt> lines = load 'anupam/comment' as (line:chararray);

grunt> dump lines;

(i love india)

(i love hadoop)

(hadoop does not love)

grunt> store lines into 'hdfs1';

[training@localhost ~]$ hadoop fs -ls hdfs1

Found 2 items

drwxr-xr-x - training supergroup 0 2015-03-28 19:43 /user/training/hdfs1/_logs

-rw-r--r-- 1 training supergroup 48 2015-03-28 19:43 /user/training/hdfs1/part-m-00000

[training@localhost ~]$ hadoop fs -cat hdfs1/part-m-00000

i love india

i love hadoop

hadoop does not love

-- hdfs mode is for production.

-- it takes input from hdfs file and writes into hdfs.

Pig Terminology:

---

i)Relation

ii)Bag

iii)Tuple

iV) Field

Field: is a data entity.

--piece of data record.

ex: ecode, name, sal, age, city.....

Tuple: Collection of feild values formed as a tuple.

ex: (101,ravi,40000,m,11)

Bag: Collection of tuples formed as a Bag.

Bag are two types:

a) Outerbag

b) Innerbag.

collection of all tuples of a dataset is called Outer bag.

emp

(101,ravi,10000,m,11)

(102,rani,30000,f,12)

(103,ram,40000,m,13)

(104,gita,50000,f,11)

Outer bag should be referenced by "Relation" name.

Innerbag: a bag placed as a field.

grp

(f,{(102,rani,30000,f,12),(104,gita,50000,f,11)})

(m,{(101,ravi,10000,m,11),(103,ram,40000,m,13)})

in above ex,

first field is chararray,

2nd field is a bag.... is called innerbag.

tuples are enclosed by ()

inner bag enclosed by {}

Pig Latin Operators:

---

1)load *

2)foreach *

3)sample *

4)limit *

5)filter *

6)dump *

7)store *

8)describe *

9)illustrate *

10)union *

11)join *

12)left outer join *

13)right outer join *

14)full outer join *

15)cross *

16)group *

17)cogroup *

18)exec *

19)run *

20)pig *

21)register

22)define

23)order *

24)distinct *

Load:- to load data from file to relation.

load has two storage methods.

i)PigStorage() -- TextInputFormat.

ii)BinStorage() -- SequenceInputFormat(BinaryFiles)

default storage method is PigStorage()

for the PigStorage() default delimiter is '\t'

ex:

grunt> cat anupam/emp

101,vino,26000,m,11

102,Sri,25000,f,11

```
103,mohan,13000,m,13

104,lokitha,8000,f,12

105,naga,6000,m,13

101,janaki,10000,f,12

grunt> emp = load 'anupam/emp' using PigStorage(',') as (ecode:int,

>>  ename:chararray, esal:int, sex:chararray, dno:int);

grunt> describe emp;

emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}

grunt>dump emp

(101,vino,26000,m,11)

(102,Sri,25000,f,11)

(103,mohan,13000,m,13)

(104,lokitha,8000,f,12)

(105,naga,6000,m,13)

(101,janaki,10000,f,12)

grunt> cat anupam/samp

100    200    500

12     23     56

1      3      5

grunt> s = load 'anupam/samp' using PigStorage('\t') as

>>    (a:int, b:int, c:int);

grunt> s2 = load 'anupam/samp' using PigStorage() as

>>    (a:int, b:int, c:int);

grunt> s3 = load 'anupam/samp'  as

>>    (a:int, b:int, c:int);
```

-- the output of s, s2 , s3 is same.

grunt> dump s

(100,200,500)

(12,23,56)

(1,3,5)

grunt> dump s2

(100,200,500)

(12,23,56)

(1,3,5)

grunt>dump s3

(100,200,500)

(12,23,56)

(1,3,5)

grunt> s4 = load 'anupam/samp' as

>> (a:int, b:int, c:int,d:int,e:int);

grunt> dump s4;

(100,200,500,,)

(12,23,56,,)

(1,3,5,,)

-- bcoz, file has only 3 fields, so d,e are nulls

grunt> s5 = load 'anupam/samp' as

>> (a:int, b:int);

grunt> dump s5

(100,200)

(12,23)

(1,3)

-- 3rd field skipped from file.

---

dump:

---

to execute entire flow of the current relation.

but output will be written on to console (grunt shell).

```
grunt> describe emp;
emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}
grunt> e2 = foreach emp generate ename, esal;
grunt> e3 = group e2 all;
grunt> res = foreach e3 generate SUM(e2.esal) as tot;
grunt> dump res
(88000)
```

store:

---

to execute entire flow of the current relation.

but output will be written into file

```
grunt> grp = group emp by sex
>> ;
grunt> r = foreach grp generate group as sex, SUM(emp.esal) as tot;
grunt> store r into 'hdfs2';
-- hdfs2 is a directory.
grunt> cat hdfs2/part-r-00000
f      43000
```

m 45000

-- here tab is the delimiter.

-- PigStorage() can be applied with load and store operators.

grunt> store r into 'hdfs3' using PigStorage(',');

grunt> cat hdfs3/part-r-00000

f,43000

m,45000

---

describe :

to get schema of the relation.

grunt> describe r;

r: {sex: chararray,tot: long}

---

illustrate:

to get entire heirarchy of the flow, with schema and sample data.

| emp | ecode: bytearray | ename: bytearray | esal: bytearray | sex: bytearray | dno: bytearray |

| | 101 | vino | 26000 | m | 11 |

| | 105 | naga | 6000 | m | 13 |

| emp | ecode: int | ename: chararray | esal: int | sex: chararray | dno: int |

| | 101 | vino | 26000 | m | 11 |

| | 105 | naga | 6000 | m | 13 |

| grp | group: chararray | emp: bag({ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}) |

| | m | {(101, ..., 11), (105, ..., 13)} |

| r | sex: chararray | tot: long |

| | m | 32000 |

-- illustrate used as debugging .

Sample:

---

to get random sample of give percentage.

grunt> s = sample emp 0.5;

grunt> dump s

(102,Sri,25000,f,11)

(103,mohan,13000,m,13)

(105,naga,6000,m,13)

limit:

---

to get top n number of tuples.

grunt> top3 = limit emp 3;

grunt> dump top3

(101,vino,26000,m,11)

(102,Sri,25000,f,11)

(103,mohan,13000,m,13)

filter:

---

to fetch matching tuples with given criteria.

grunt> f1 = filter emp by (sex=='f')

>> ;

grunt> f2 = filter emp by (sex=='f' and esal>=20000);

---

order:

to sort the data either in ascending or desceding order.

grunt> o1 = order emp by ename;

grunt> o2 = order emp by esal desc;

grunt> o3 = order emp by esal desc, dno , sex desc;

grunt> dump o3

(101,vino,26000,m,11)

(102,Sri,25000,f,11)

(103,mohan,13000,m,13)

(101,janaki,10000,f,12)

(104,lokitha,8000,f,12)

(105,naga,6000,m,13)

---

foreach:

-- copy

-- generate new fields

-- filter fields

-- change field order

-- renaming fields

-- type casting

-- transformations.

--copy:

```
grunt> e = foreach emp generate *;
```

--filter fields:

```
grunt> e2 = foreach emp generate ename,esal;
```

--generating new fields:

```
grunt> e3 = foreach emp generate *, esal*0.1 as tax,
>>          esal*0.2 as hra;
grunt> e3 = foreach e3 generate *, esal+hra-tax as net;
```

--changing field order:

```
grunt> e4 = foreach e3 generate ecode, ename, esal, tax, hra, net,
>>          dno, sex;
grunt> describe e4
2015-03-28 21:08:52,423 [main] WARN  org.apache.pig.PigServer - Encountered Warning
IMPLICIT_CAST_TO_DOUBLE 4 time(s).
e4: {ecode: int,ename: chararray,esal: int,tax: double,hra: double,net: double,dno: int,sex: chararray}
```

--you can call fields using Index numbers.

```
grunt> x = foreach e3 generate $2 as salary;
```

--rename fields

grunt> e5 = foreach emp generate ecode, ename as name, esal,

>> sex as gender, dno ;

grunt> describe e5;

e5: {ecode: int,name: chararray,esal: int,gender: chararray,dno: int}

--type casting:

grunt> describe e4;

2015-03-28 21:12:20,787 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 4 time(s).

e4: {ecode: int,ename: chararray,esal: int,tax: double,hra: double,net: double,dno: int,sex: chararray}

grunt> e6 = foreach e4 generate ecode,ename,esal,(int)tax,

>> (int)hra,(int)net;

grunt> describe e6;

2015-03-28 21:13:11,520 [main] WARN org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 4 time(s).

e6: {ecode: int,ename: chararray,esal: int,tax: int,hra: int,net: int}

--transformations:

grunt> cat anupam/emp

101,vino,26000,m,11

102,Sri,25000,f,11

103,mohan,13000,m,13

104,lokitha,8000,f,12

105,naga,6000,rn,13

101,janaki,10000,f,12

grunt> describe emp

emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}

```
grunt> e7 = foreach emp generate ecode, ename, esal,
>>    (esal>=20000 ? 'A':
>>        (esal>=14000 ? 'B':
>>          (esal>=9000 ? 'C':'D'))) as grade,
>>    (sex=='f' ? 'Female':'Male') as sex,
>>    (dno==11 ? 'Marketing':
>>        (dno==12 ? 'Hr':
>>          (dno==13 ? 'Finance':'Others'))) as dname;
grunt> describe e7
e7: {ecode: int,ename: chararray,esal: int,grade: chararray,sex: chararray,dname: chararray}
grunt> store e7 into 'hdfs4';
grunt> cat hdfs4/part-m-00000
101    vino    26000    A      Male    Marketing
102    Sri     25000    A      Female  Marketing
103    mohan   13000    C      Male    Finance
104    lokitha 8000     D      Female  Hr
105    naga    6000     D      Male    Finance
101    janaki  10000    C      Female  Hr
--cleaning nulls:
grunt> cat anupam/sampx
100,200
,200
300,
100,300
```

```
grunt> s = load 'anupam/sampx' using PigStorage(',') as
>>      (a:int, b:int);
grunt>dump s
(100,200)
(,200)
(300,)
(100,300)
grunt> r = foreach s generate *,a+b as tot;
grunt> dump r
(100,200,300)
(,200,)
(300,,)
(100,300,400)
grunt> s = foreach s generate (a is null ? 0:a) as a,
>>          (b is null ? 0:b) as b;
grunt> r = foreach s generate *,a+b as tot;
grunt> dump r
(100,200,300)
(0,200,200)
(300,0,300)
(100,300,400)
```

Union:

merging datasets

```
grunt> cat anu/emp
```

```
101,vino,26000,m,11

102,Sri,25000,f,11

103,mohan,13000,m,13

104,lokitha,8000,f,12

105,naga,6000,m,13

101,janaki,10000,f,12

107,kkk,30000,m,15

108,kkdk,40000,f,20

grunt> cat anu/emp2

201,kkdkdkd,90000,m,12

101,vino,26000,m,11

101,vino,26000,m,11

101,vino,26000,m,11

grunt> cat anu/emp3

301,kd,11,m,20000

302,kj,12,f,3000

grunt> emp = load 'anu/emp' using PigStorage(',')

>>   as (ecode:int, ename:chararray, esal:int, sex:chararray,

>>       dno:int);

grunt> emp2 = load 'anu/emp2' using PigStorage(',')

>>   as (ecode:int, ename:chararray, esal:int, sex:chararray,

>>       dno:int);

grunt> emp3 = load 'anu/emp3' using PigStorage(',')

>>   as (ecode:int, ename:chararray, dno:int, sex:chararray, esal:int);

grunt> emp3 = foreach emp3 generate ecode, ename, esal, sex, dno;
```

grunt> e = union emp, emp2, emp3;

grunt> dump e

(101,vino,26000,m,11)

(102,Sri,25000,f,11)

(103,mohan,13000,m,13)

(104,lokitha,8000,f,12)

(105,naga,6000,m,13)

(101,janaki,10000,f,12)

(107,kkk,30000,m,15)

(108,kkdk,40000,f,20)

(201,kkdkdkd,90000,m,12)

(101,vino,26000,m,11)

(101,vino,26000,m,11)

(101,vino,26000,m,11)

(301,kd,20000,m,11)

(302,kj,3000,f,12)

-- pig union is equivalant to sql union all

-- it allows duplicates

Distinct:-

to eliminate duplicates:

grunt> e = distinct e;

--based on entire tuple match , duplicates will be eliminated.

--to list out unique department numbers:

grunt> e = distinct e;

grunt> x = foreach e generate dno;

```
grunt> y = distinct x;

grunt> dump y

(11)

(12)

(13)

(15)

(20)
```

Aggregations:

```
grunt> r1 = foreach emp generate SUM(esal) as tot;

grunt> dump r1

2015-03-28 21:38:35,859 [main] ERROR org.apache.pig.tools.grunt.Grunt - ERROR 1045: Could not infer the matching function for org.apache.pig.builtin.SUM as multiple or none of them fit. Please use an explicit cast.

Details at logfile: /home/training/pig_1427598209145.log

grunt>

-- Note: aggregations should be applied on only inner bags.

 -- when you group the data , inner bags will be produced.

--Performing Column aggregations:
```

```
 -- all tuples making as one group.

grunt> e = foreach emp generate esal;

grunt> grp = group e all;

grunt> dump grp

(all,{(26000),(25000),(13000),(8000),(6000),(10000),(30000),(40000)})
```

```
grunt> describe grp

grp: {group: chararray,e: {esal: int}}

grunt> rsum = foreach grp generate SUM(e.esal) as tot;

grunt> describe rsum

rsum: {tot: long}

grunt> dump rsum

(158000)

-- all aggregations of a column

grunt> rall = foreach grp generate SUM(e.esal) as tot,
>>       AVG(e.esal) as avg,
>>       MAX(e.esal) as max,
>>       MIN(e.esal) as min,
>>      COUNT(e) as cnt;

grunt> describe rall

rall: {tot: long,avg: double,max: int,min: int,cnt: long}

grunt> rall = foreach rall generate (int)tot,(int)avg,max,min,
>>           (int)cnt;

grunt> dump rall

(158000,19750,40000,6000,8)

-- performing grouping aggregations:

grunt> es = foreach emp generate sex, esal;

grunt> bySex = group es by sex;

grunt> dump bySex

(f,{(f,25000),(f,8000),(f,10000),(f,40000)})

(m,{(m,26000),(m,13000),(m,6000),(m,30000)})
```

```
grunt> sexsum = foreach bySex generate group as sex,
>>          SUM(es.esal) as tot;
grunt> store sexsum into 'hdfs5';
grunt> cat hdfs5/part-r-00000
f     83000
m     75000
-- performing multi grouping aggregations.
 -- pig does not allow multi grouping.
  -- make mulitple fields as a tuple, group it by tuple.
grunt> grp = group emp by (dno,sex);
grunt> describe grp;
grp: {group: (dno: int,sex: chararray),emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}}
grunt> res = foreach grp generate group.dno as dno, group.sex as sex,
>>       SUM(emp.esal) as tot;
grunt> store res into 'hdfs6';
grunt> cat hdfs6/part-r-00000
11    f     25000
11    m     26000
12    f     18000
13    m     19000
-- performing grouping aggregations seperately foreach dataset.
  describe emp;
 emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}
 emp2 = load 'anu/emp2' using PigStorage(',')
 >>  as (ecode:int, ename:chararray, esal:int, sex:chararray,
```

```
>>        dno:int);

emp3 = load 'anu/emp3' using PigStorage(',')

>> as (ecode:int, ename:chararray, dno:int, sex:chararray,

>>        esal:int);

grunt> cg = cogroup emp by sex, emp2 by sex, emp3 by sex;

grunt> describe cg

cg: {group: chararray,emp: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int},emp2: {ecode:
int,ename: chararray,esal: int,sex: chararray,dno: int},emp3: {ecode: int,ename: chararray,dno: int,sex:
chararray,esal: int}}

grunt> dump cg

(f,{(102,Sri,25000,f,11),(104,lokitha,8000,f,12),(101,janaki,10000,f,12)},{(203,nnn,34000,f,12)},{(301,hg,1
1,f,50000)})

(m,{(101,vino,26000,m,11),(103,mohan,13000,m,13),(105,naga,6000,m,13)},{(202,kkkk,40000,m,12),(20
3,ccccc,40000,m,11),(204,dddd,50000,m,12)},{(302,kdkdkd,12,m,40000),(303,iiu,11,m,50000)})

grunt> res = foreach cg generate group as sex,

>>        SUM(emp.esal) as tot1, SUM(emp2.esal) as tot2,

>>          SUM(emp3.esal) as tot3;

grunt> store res into 'hdfs7';

grunt> cat hdfs7/part-r-00000

f      43000   34000   50000

m      45000   130000  90000
```

Submitting scripts:

```
[training@localhost ~]$ cat anu1.pl

emp1 = load 'anu/emp' using PigStorage(',')

 as (ecode:int, ename:chararray, esal:int, sex:chararray, dno:int);

emp2 = load 'anu/emp2' using PigStorage(',')
```

as (ecode:int, ename:chararray, esal:int, sex:chararray, dno:int);

emp3 = load 'anu/emp3' using PigStorage(',')

as (ecode:int, ename:chararray, dno:int, sex:chararray, esal:int);

emp4 = foreach emp3 generate ecode, ename, esal, sex, dno;

e = union emp1, emp2, emp4;

dump e;

[training@localhost ~]$ pig anu1.pl <enter>

note: relation aliases not available in grunt.

We can also excute scripts from grunt shell.

grunt> exec anu1.pl

-- still relation aliases not available in grunt.

grunt> run anu1.pl

-- now relation aliases available.

---

grunt> cat anu/emp

101,vino,26000,m,11

102,Sri,25000,f,11

103,mohan,13000,m,13

104,lokitha,8000,f,12

105,naga,6000,m,13

101,janaki,10000,f,12

107,kkk,30000,m,15

108,kkdk,40000,f,20

grunt> cat anu/dept

11,marketing,hyd

12,hr,del

13,finance,del

14,prod,hyd

grunt> describe emp1

emp1: {ecode: int,ename: chararray,esal: int,sex: chararray,dno: int}

grunt> dept = load 'anu/dept' using PigStorage(',') as

>> (dno:int, dname:chararray, dloc:chararray);

Inner Join:

grunt> ij = join emp1 by dno, dept by dno;

left outer join:

grunt> lj = join emp1 by dno left outer, dept by dno;

right outer join:

grunt> rj = join emp1 by dno right outer, dept by dno;

full outer join

grunt> fj = join emp1 by dno full outer, dept by dno;

grunt> dump fj

(101,vino,26000,m,11,11,marketing,hyd)

(102,Sri,25000,f,11,11,marketing,hyd)

(104,lokitha,8000,f,12,12,hr,del)

(101,janaki,10000,f,12,12,hr,del)

(103,mohan,13000,m,13,13,finance,del)

(105,naga,6000,m,13,13,finance,del)

(,,,,,14,prod,hyd)

(107,kkk,30000,m,15,,,)

(108,kkdk,40000,f,20,,,)

```
grunt> describe ij

ij: {emp1::ecode: int,emp1::ename: chararray,emp1::esal: int,emp1::sex: chararray,emp1::dno:
int,dept::dno: int,dept::dname: chararray,dept::dloc: chararray}

grunt> j = foreach ij generate dept::dloc as city, emp1::esal as sal;

grunt> describe j;

j: {city: chararray,sal: int}

grunt> grp = group j by city;

grunt> res = foreach grp generate group as city, SUM(j.sal) as tot;

grunt> store res into 'hdfs8';

grunt> cat hdfs8/part-r-00000

del    37000

hyd    51000
```

```
grunt> info = foreach fj generate emp::esal as sal,
>>          emp::dno as dno1, dept::dno as dno2;
grunt> info2 = foreach info generate sal,
>>          (dno1 is not null and dno2 is not null ? 'Working':
>>          (dno1 is null ? 'BenchProj':'BenchTeam')) as stat;
grunt> grp = group info2 by stat;
grunt> res = foreach grp generate group as stat, SUM(info2.sal) as tot,
>>          COUNT(info2) as cnt;
grunt> dump res;
(Working,88000,6)
(BenchProj,,0)
(BenchTeam,70000,2)
```

Cross: to get cartisian product.

```
grunt> e1 = foreach emp1 generate ecode, esal;
grunt> e2 = foreach e1 generate *;
grunt> cr = cross e1, e2;
grunt> describe cr;
cr: {e1::ecode: int,e1::esal: int,e2::ecode: int,e2::esal: int}
grunt> crr = foreach cr generate e1::ecode as ecode,
>>          e1::esal as sal1 , e2::esal as sal2;
grunt> crrr = filter crr by (sal1>sal2);
grunt> grp = group crrr by ecode;
grunt> res = foreach grp generate group as ecode, COUNT(crrr) as cnt;
grunt> store res into 'hdfs9';
grunt> cat hdfs9/part-r-00000
101    7
102    4
103    3
104    1
107    6
108    7
```

Pig UDF:-(user defined function).

---

we need to configure "pig-core.jar"

Pig UDF classes:

i) org.apache.pig.EvalFunc

ii) org.apache.pig.data.Tuple

whenever your java class extended EvalFunc,

then the class will get UDF functionality.

The UDF logic, to be overwridden in exec()

exec() is called for n times, where n is number of input tuples of pig relation.

exec() has an argument of type "Tuple"

exec(Tuple v)

whatever arguments you have passed the values will be stored in tuple variable.

tuple variable can contain multiple fields.

to get first field.

v.get(0)

v.get(1)

v.get(9)

ex udf, to find max value of row.(row level max)

(ctrl +shift + o) – this to import classes

step1)

```
package pig.test;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
public class RowMax extends EvalFunc<Integer>
{
        public  Integer   exec(Tuple v)
```

```
        throws IOException
        {
                int a =(Integer) v.get(0);  //10
                int b =(Integer) v.get(1);  //23
                int c=(Integer) v.get(2);   //19
                int big=0;
                if(a>big) big=a;
                if(b>big) big=b;
                if(c>big) big=c;

                return new Integer(big);
        }

}
```


step2) Export into jar file.

project(AnuDemo)—export —java —java jar—path of jar

ex: /home/training/Desktop/anu.jar

step3) Register the jar file into pig.

grunt> register Desktop/anu.jar;

step4) create temporary function for the UDF classs.

grunt> define rmax pig.test.RowMax();

step5) calling the function.

grunt> cat anupam/samp

100 200 500

12 23 56

```
1    3    5

grunt> s = load 'anupam/samp' as (a:int, b:int, c:int);

grunt> res = foreach s generate *, rmax(*) as max;

grunt> describe res

res: {a: int,b: int,c: int,max: int}

grunt> store res into  'hdfs10';

grunt> ls hdfs10

hdfs://localhost/user/training/hdfs10/_logs    <dir>

hdfs://localhost/user/training/hdfs10/part-m-00000<r 1> 36

grunt> cat hdfs10/part-m-00000

100    200    500    500

12     23     56     56

1      3      5      5
```

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678

Hive

Hive is a data warehouse environment in hadoop framework.

where you maintain data in tables in structured format.

Hive runs on top of hdfs and MapReduce. that means for hive, the backend storage is hdfs and executim model is MapReduce.

→ when you create hive table, In hdfs, with table name one directory will be created.

→ when you load a file into table, the file will be copied into table's backend hdfs directory.

this means, hive gives a table shape (structured shape) for the hdfs file.

→ when you submit a query (hql), the statement is converted as MapReduce, converted code will be submitted to JVM, Because hadoop runs on JVM.

So finally for the hive, executim model is MapReduce

HQL (Hive Query language) is used to manage and process data.

HQL can process structured data,
XML,
Json,
URIs [Major Content of Weblogs].

but hive is weak for unstructured text data to process.

hql is similar to sql of RDBMS.

But hql is not for operating rows randomly.

Such as reading a record randomly,
Insert/update/delete records randomly.

→ In older versions (such as 0.7.*, 0.8.*..) of hive, hive does not support update and delete operations.
But latest versions (0.14.*..), hive supports update and delete.

→ Hive supports only <u>sequential Reading.</u>
↳ Record by Record Reading (one after one) from Block begin to Block ending.

Hive DDL (Data Definition Language)

→ Create
→ Alter
→ Drop.

In hive,
Each table should be associated with a database.

A database is a group (logical) of tables, views.

The default database in hive is "default".

If you don't select any database, your table will use "default" database.

Following, is the hadbs location of "default" database

~~/user/training/~~
/user/hive/warehouse

When table is created in hive default database, a directory will be created in warehouse directory.

ex:-
/user/hive/warehouse/tab1
→ directory of hive table tab1.

We can also create custom databases (user defined).

Enter into hive shell (hive Console) :-

```
$ hive ↵
hive>
```

get list of existed databases :-

```
hive> show ~~tabl~~ databases; ↵
=
default → default data base
Sales } → user defined database.
accounts }
=
```

How to select a database :-

```
hive> use <db name>;
```

ex:-

```
hive> use sales;
```

If any table is created, the table will be associated with sales database.

and table directory will be created under sales.db directory of hdfs.

How to create a database :-

```
hive> create database mydb;
```

In hdfs, a directory will be created in warehouse location, with .db extension.

ex:-

In HDFS,

/user/hive/warehouse/mydb.db
└→ Directory.

Now if a table created under mydb database a sub directory will be created under mydb.db in hdfs.

Let us see following examples.,

(i)

```
hive> use default;
hive> create table tab1 (line string);
```

In HDFS,

/user/hive/warehouse/tab1

(ii)

```
hive> use mydb;
hive> create table tab2 (line string);
```

In HDFS,

/user/hive/warehouse/mydb.db/tab2

How to drop a ~~table~~ database :—

```
hive> drop database mydb;
```

NOTE:- Before dropping a database, the database should be empty.

Hive Tables.

Hive Tables can be classified in following ways.

A) Inner & External Tables.

B) partitioned and non-partitioned tables

C) Bucketed & non-Bucketed tables.

First we see Inner and External table.

Inner	External
→ When an inner table is dropped, both metadata and data will be deleted.	→ When an external table is dropped, only metadata will be deleted.
→ From hive, table is dropped, From hdfs, table directory will be ~~developed~~ deleted	→ From hive, table is dropped, But In hdfs, table directory available.
→ So you lost ~~only~~ table and data.	→ So you only lost table but not data. you can reuse data for further.

Let's see examples of inner & external tables

```
$cat > file1
AAAA
AAAA
^d
```
→ local file

Inner table (By default each table is inner table)

```
hive> create table tab1(line string);
```

In HDFS,

/user/hive/warehouse/tab1 → directory is created.

-- loading file into table.

```
hive> load data local inpth 'file1' into table tab1;
```

In HDFS,

/user/hive/warehouse/tab1/file1

→ file1 is copied into tab1 directory from local.

```
hive> selçit * from tab1;
AAAA
AAAA
```

Now hive starts reading data from all files of table-directory.

```
$cat > file2
BBBB
BBBB
^d → (for save)
hive > load data local inpath 'file2' into table tab1;
```

$hadoop fs -ls /user/hive/warehouse/tab1 ↵

→ /user/hive/warehouse/tab1/file1
/user/hive/warehouse/tab1/file2

Now in hdfs, tab1 directory has two files file1 and file2.

hive> select * from tab1;

Now hive reads all rows of file1 and file2.

-- dropping a table.

hive> drop table tab1;

now from hive, tab1 will be deleted.

In HDFS

/user/hive/warehouse/tab1 directory will be removed. (file1, file2 deleted).

[Hive maintains metadata (Hive Metastore) in RDBMS. table & its backend HDFS location will be maintained in metastore, Later we discuss more about this metadata]

Now tab1 (table) and data (HDFS directory) both deleted.

This behavior of inner tables

Generally temporary data or intermediate data of your data process; we keep into inner tables.

Hive External tables.

By Mr. SREERAM
(Data Scientist)

Creating an External tables :—

```
hive> Create External table eTab (line string);
```

In HDFS,

/user/hive/warehouse/etab

```
hive> load data local inpath 'file1' into table eTab;
```

In HDFS,

/user/hive/warehouse/etab/file1

```
hive> drop table eTab;
```

From Hive, eTab is deleted [Metadata deleted]

In HDFS, /user/hive/warehouse/eTab/file1 is available.
(Data)

Here, only table is deleted, not the data.

Reuse the data

```
hive> Create table eTab( line string);
```

Now, eTab directly to be created in warehouse,
But eTab is already existed. So hive will
Reuse the location /user/hive/warehouse/eTab/file1

```
hive> select * from eTab;
```
→ (As Inner table)

AAAA
AAAA

with out loading data available.

But now, (previous) we created eTab as inner table.

if you drop eTab,

Now both table and directory will be deleted.

Both inner and external tables can use Custom hdfs locations.

ex:

→ Inner

hive> create table tabx(line string)
location '/user/myloc';

In hive, tabx created

In hdfs, /user/myloc directory created.

hive> load data local inpath 'file1' into table tabx;

In HDFS, /user/myloc/file1

Now table name is tabx,
directory name is myloc.

hive> drop table tabx;

user/myloc will be deleted.
because tabx is inner table.

External table can also use custom location.

ex:-

~~hive> create table~~

hive> create external table taby(line string)
location '/user/urtab';

As already we know, if urtab is existed, hive will use it, if not existed ~~hive~~ a directory will be created with given name.

hive> drop table taby;

Still /user/urtab is available. (because taby is external table).

LOADING DATA INTO TABLES

When you loaded a file in to table the file will be copied into table's HDFS Directory.

ex:-

```
$ cat > samp1
1
2
3
^d
$ cat > samp2
4
5
6
^d
```

LOADING FROM LOCAL FILES

```
hive> Create table samp(line string);
hive> load data local inpath 'samp1' into
                        table samp;
hive> load data local inpath 'Samp2' into
                        table samp;
```

NOW: In hdfs, Samp directory has two files.
- samp1 and samp2.

that means, foievery load, file is copied into table directory. So data is appending.

Another way to load data into table

```
$ cat > Samp3
7
8
9
^d
$ hadoop fs -copyFromLocal samp3 \
        /user/hive/warehouse/samp ↵
```

now Samp directory has 3 files file1, file2 and file3.

```
hive> select * from samp;
```

op →
```
1
2
3      } from samp1
4
5      } from Samp2
6
7
8      } from Samp3.
9
```

How to overwrite data of table, while loading :—

```
$ cat > samp4
10
11
12
^d
```

```
hive> load data local inpath 'samp4' overwrite
    into table samp;
```

Now all previous files of samp directory, samp1 samp2 and samp3 will be deleted. and samp4 file is copied into samp.

```
hive> select * from samp;
10
11
12
```

```
hive> load data local inpath 'samp4'
    into table samp;
```

→ is already existed in samp.

```
$ hadoop fs -ls /user/hive/warehouse/samp ↵
    ...samp4
    ...samp4_copy_1
hive> select * from samp;
10
11
12      } from samp4
10
11
12      } from samp4_copy_1
```

loading data from HDFS to hive table.

```
hive> load data inpath '/user/training/file1'
        into table Samp;
```

→ once file1 is loaded into table, source file (file1) will be deleted from hdfs.

In all previous examples we worked with single column tables.

What if, if file has multiple ~~tables~~ fields, now we need to create table with multiple columns.

ex

```
$cat > test1
100, 200, 300
400, 500, 600
700, 800, 900
^d
hive> Create table Test1(a int, b int, c int);
hive> load data local inpath 'test1' into table Test1;
hive> select * from Test1;
Null  Null  Null
Null  Null  Null
Null  Null  Null
```

} Why?

↓

The problem is with delimiter

Default delimiter for hive table is "♦" (\001)

But Input file has "," [comma] delimiter.

$ cat test1

100, 200, 300

⋮

for the table 'Test1' delimiter is ◇ (\001).

But file has ∅ delimiters. (only one field)

so entire line is treated as single field.

total line to be stored in first column of the table (a int).

First column in Test1 table is int, but line is alphanumeric.

so column 'a' became null.

there is no 2nd field and 3rd field,

thats why 'b', 'c' columns became null in previous example.

Solution:- modify the delimiter with ',' (comma). (overwrite)

hive> create table Test2(a int, b int, c int)
row format delimited fields terminated by ',';

now for the table 'Test2', ',' is delimiter,

file 'test1' has two commas,

so number of fields in file is 2+1 = 3.

1st field will be assigned to 'a' column.

2nd " " " 'b' " .

3rd field " " 'c' " .

hive> load data local inpath 'test1' into table Test2;

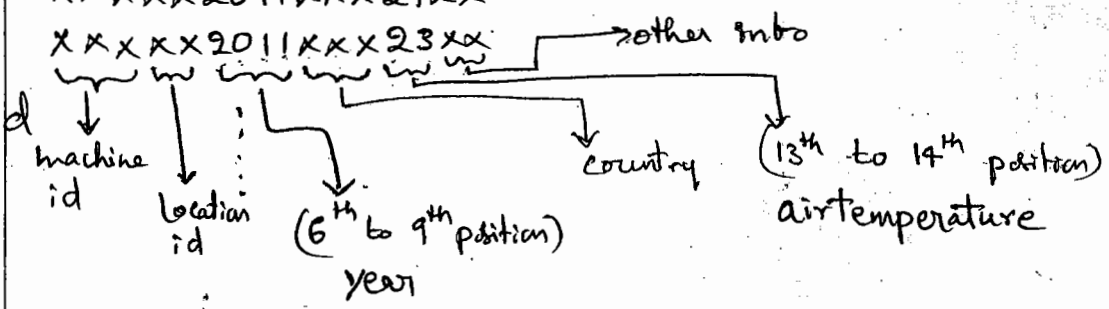
hive> select * from Test2;

100	200	300
400	500	600
700	800	900

Let us work with temperature data using hive:-

$ cat > temperature.txt

```
XXXXX2010XXX20XX
XXXXX2010XXX21XX
XXXXX2010XXX20XX
XXXXX2011XXX27XX
XXXXX2011XXX23XX
```

- this file has no delimiters.

But we have clear clues for attributes.

year → 6th to 9th

temperature → 13th to 14th.

Task:- for each year, max temperature.

```
hive> create table Rawtemperature(line string);
hive> load data local inpath 'temperature.txt'
      into table Rawtemperature;
-- we will see how to load data from one table to
-- another table.
hive> create table tmpr(y int, t int);
hive> insert overwrite table tmpr
      select substr(line,6,4), substr(line,13,2)
      from Rawtemperature;
```

→ for year → for tmpr

```
hive> Select * from tmpr;

2010  20
2010  21
2010  20
2011  27
2011  23
```


Substr(line, 6, 4) → will extract year.

(6 → start position, 4 → length of substring)

- Substr(line, 13, 2) → will extract temperature.

```
hive> Create table Result(y int, max int);
hive> Insert overwrite table Result
        select y, max(t) from tmpr
            group by y;
hive> select * from Result;
```

o/p →

```
2010  21
2011  27
```

```
$ hadoop fs -ls /user/hive/warehouse/Result'
  → /user/hive/warehouse/Result/000000_0
```

↳ When table to table load is done, hive creates a data file with name 000000_0.

```
$ hadoop fs -cat /user/hive/warehouse/Result/000000_0
```

(use small r)

```
2010◊21
2011◊27
```

↳ \001

In previous examples we used

"Insert overwrite table <table name>"

↳ is mandatory.

but hive 0.14, "Insert Into" available to append rows.

"Insert overwrite" will overwrite previous data.

One more example on temperature data, with air temperature has both positive and negative values.

$ cat > tmpr.txt

```
xxxxx2010xxx20xx
xxxxx2010xxx-15xx
xxxxx2010xxx-10xx
xxxxx2010xxx23xx
xxxxx2011xxx-15xx
xxxxx2011xxx-11xx
```

This file has both negative and positive values.

now for temperature,

if positive, length is 2.

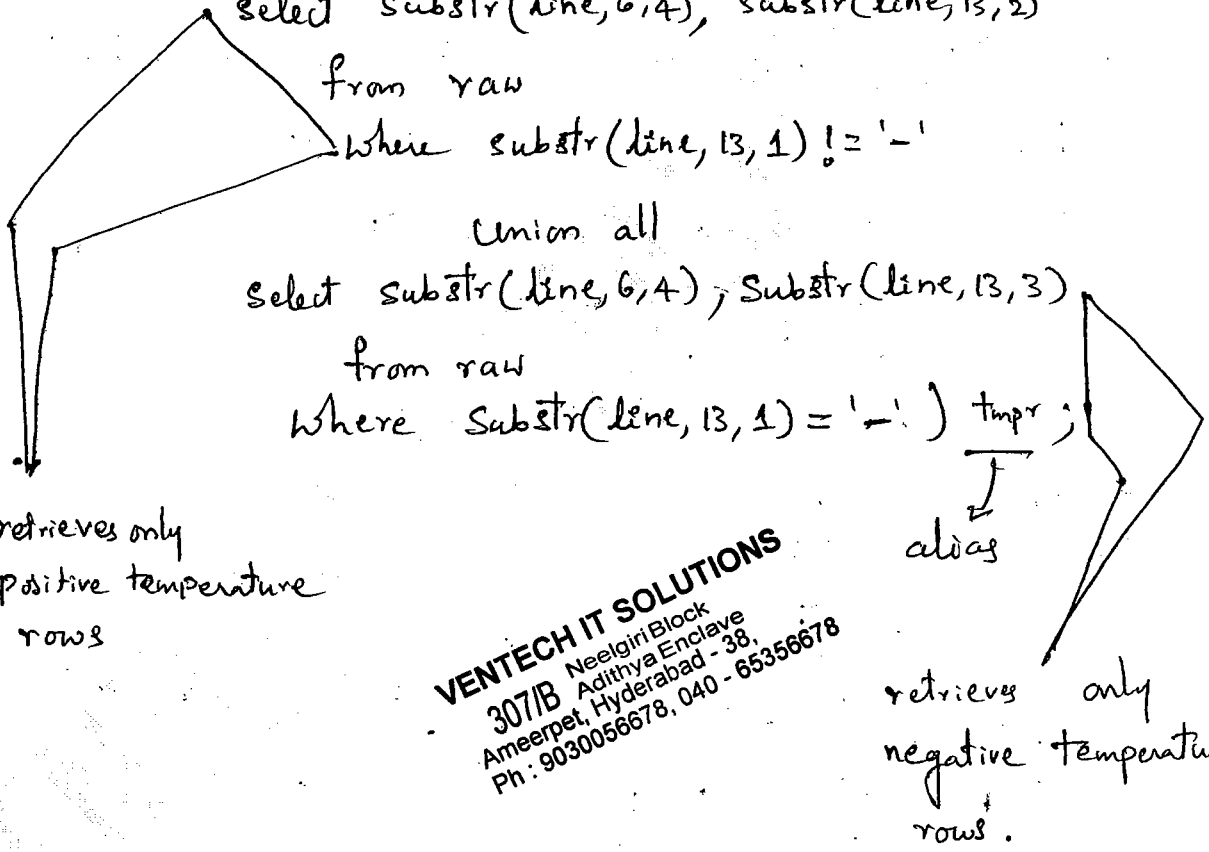
if negative, length is 3.

How to make it structure, and analyse it.?

Task :- for each year, max and min temperature is required.

```
hive> create table raw(line string);
hive> load data local inpath 'tmpr.txt' into table raw;

hive> Create table tempr(y int, t int);
hive> Insert overwrite table tempr
        select * from (
          select substr(line,6,4), substr(line,13,2)
            from raw
            where substr(line,13,1) != '-'
              Union all
          select substr(line,6,4), substr(line,13,3)
            from raw
            where substr(line,13,1) = '-' ) tmpr ;
```



Using Union All both rows set we merged.

→ In hive there is NO "Union", only "Union All" available.

→ "Union All" should be applied as a sub query.

→ The sub query should have an alias. (ex:- tmpr)

Merged Results we are loading in tempr table.

```
hive> select * from tempr;
```

o/p →

2010	20
2010	-15
2010	-10
2010	23
2011	-15
2011	-11

```
hive> create table Result2(y int, max int, min int)
      row format delimited fields terminated by ',';
hive> Insert overwrite table Result2
      select y, max(t), min(t) from tempr
        group by y;
hive> select * from Result2;
```

o/p →

2010	23	-15
2011	-11	-15

```
$ hadoop fs -cat ~~Resta~~ /user/hive/warehouse/result2/000000_0
2010,23,-15
2011,-11,-15
```

} because we set comma delimiter for table.

Hive Complex datatypes & Simple Data types

Hive Simple Data types

Tinyint → (1-byte Signed integer) → -128 to 127

Smallint (2 byte Signed integer) → -32,768 to 32,767

int (4 byte Signed integer) → -2,147,483,648 to 2,147,483,647

Bigint (8 byte Signed integer) → -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

float (4 byte Single pecasion floating point Number)

double (8 byte double " " ")

DECIMAL (Available from 0.11.0 version onwards)

→ In hive 0.11.0, with a precision of 38 digits.

→ From hive 0.13.0 onwards, user definable precision and scale available.

Date (Available from 0.12.0 onwards)

Timestamp (available from 0.8.0 onwards)

String (Max size 2GB)

VARCHAR (available from 0.12.0 onwards)

1 to 65355 characters.

CHAR (available from 0.13.0 onwards)

BOOLEAN → True/FALSE

BINARY → (available from 0.8.0 onwards)

Hive Complex data types.

Collection data types:-

~~→ Arrays~~

→ Array → collection of elements

→ Map → collection of key and value pairs

→ STRUCT → collection of attributes with different data types.

Working with Arrays

hive> ! cat student → to display local file content from hive

ravi, 23, btech*mtech*phd

rani, 24, bsc*msc*mtech

```
hive> create table student(name string, age int,
        qualification array<string>)
      row format delimited fields terminated by ','
        collection items terminated by '*';
```

hive> ~~select~~ load data local inpath 'student' into table student;

hive> select * from student;

ravi	23	["btech", "mtech", "phd"]
rani	24	["bsc", "msc", "mtech"]

[In later examples we will see how handle arrays more effectively]

hive> select qualification[0] from student;

btech

bsc

Working with Map

```
$ cat trans
c101, p1$1000 * p2$2000 * p3$4000
c102, p1$1200 * p4$2500 * p2$5000 * p7$6000
```

```
hive> create table trans(cid string,
        products map<string, int>)
      row format delimited
        fields terminated by ','
        Collection items terminated by '*'
        Map keys terminated by '$';
hive> load data local inpath 'trans' into table trans;
hive> select * from trans;
c101   {"p1":1000, "p2":2000, "p3":4000}
c102   {"p1":1200, "p4":2500, "p2":5000, "p7":6000}
hive> select ~~prod~~ cid, products['p2'] from trans;
```

c101	2000
c102	5000

Working with Struct

```
$ cat > prob
rasi, 24, rani#24#del, hyd
giri, 29, vani#25#hyd, hyd
^d
```

```
hive> create table probile (name string,
        age int,
      wife struct<name:string, age:int, city:string>,
      city string)
    row format delimited
      fields terminated by ';'
      Collection items terminated by '#';
hive> load data local inpath 'prob' into table probile;
hive> selet * from probile;
ravi    24    {"name":"rani", "age":24, "city":"del"}  hyd
giri    29    {"name":"vani", "age":25, "city":"hyd"}  hyd
hive> select name, wife.name from probile;
ravi    rani
giri    vani
```

Few more hql operations we will see --
With a simple structured file emp;

$ cat > emp

101, AA, 20000, M, 11
102, BBB, 30000, F, 12
.....


```
hive> create table emp(ecode int, name string,
   sal int, sex string, dno int);
    row format delimited fields terminated by ',';
```

-- to check the structure of table.

```
hive> describe emp;
hive> load data local inpath 'emp' into table emp;
```

-- Now we see different select statements.

```
hive> select * from emp;
```

→ All rows, all columns.

```
hive> select * from emp where sex='F';
```

→ All Females

```
hive> select * from emp where dno=11 and sex='M';
```

→ All Males of dno 11.

```
hive> select * from emp where
        ((dno=11 or dno=12) and sex='M')
                or
        ((dno=13 or dno=15) and sex='F')
                or
        (dno=14 or dno=20);
```

→ all Males of 11, 12
→ all females of 13, 15 and all rows of 14, 20.

```
hive> select * from emp
        where dno=11 or dno=13 or dno=14;
```

or

```
hive> select * from emp
        where dno in (11,13,14);
```

→ all rows of dno → 11,13,14.

~~Except~~

```
hive> select * from emp where dno!=13 ~~and dno!~~
        and dno!=15;
```

→ Except 13, 15 dnos, Remaining dnos will come.

In SQL, you can say

```
select * from emp where dno not in (13,15);
```

But "NOT IN" operator not available in hive.

```
hive> select * from emp where sal>=20000 and
        sal<=80000;
```

→ all rows, whose sal in between 20000 and 80000.

In SQL,

```
select * from emp where sal between 20000 and
        80000;
```

"Between" operator not available in Hive
and also "NOT BETWEEN" is not available in hive.

```
hive> seled * from emp where sal<30000 or
        sal>70000;
```

→ rows of which sal is not between,
30000 and 70000.

~~hive>Sele~~

hive> select * from emp limit 5;

→ top 5 rows of the table.

→ but we can not fetch last n number of rows.

→ Such things can be done using UDFs.

Order by

to sort the data either in ascending or descending order.

ex:-

hive> select * from emp order by name;

→ default ascending order.

hive> select * from emp order by ~~desc~~ sal desc;

↳ descending order.

hive> select * from emp order by sal desc, dno, sex desc;

↳ sort by multi columns.

It sorts the data based on ~~sdsa~~ sal in descending order. In sal high value will be given priority.

If two sals are equal, then it compares with dno. for dno, no sort mode specified, default is ascending. So low value of dno is prioritized. If two dnos are equal then it compares with sex. High value will be prioritized...

In this way, no limit for max no. of sort columns.

Adding new columns to table

```
hive> Alter table emp add columns (tax int);
hive> select * from emp;
```

					→ tax
101	AA	20000	M	11	Null
102	BBB	30000	F	12	Null

```
hive> Insert overwrite table emp
        select ecode, name, sal, sex, dno, sal*0.1
          from emp;
```

(First expression to first column, 2nd expression to 2 column ...)

In above example target table and source table is same. So data will be overriden.

```
hive> select * from emp;
```

101	AA	20000	M	11	2000
102	BBB	30000	F	12	3000

Updating a table

Task:- 10% increment to sal of each row.

Equivalent SQL query

```
update emp set sal = sal + (sal*0.1);
```

```
hive> Insert overwrite table emp
        select ecode, name, sal + (sal*0.1), sex, dno, tax
          from emp;   → source, target same.
```

By using overwriting technique we can update data.

Like

```
hive> create table staff like emp;
-- NOW, with same structure of emp a new table
   will be created called staff:
-- emp has ',' (comma) delimiter,
     So, staff is also having ',' delimiter.

hive> Insert overwrite table staff
        select * from emp;
```

---

deleting rows of a table

→ In hive 0.14 we have delete and update statements
[we see later 0.14 features in Future topics]

```
hive> -- task is to delete female rows from staff table.
    > -- But we dont have delete statement.

hive> Insert overwrite table staff
        select * from staff where sex='M';
-- All Male rows copied into staff, so indirectly
-- female rows deleted.
```

If function :—

Used to perform Caditional transformations

If (Conditition, True value, False value)

1st argument is Condition
2nd argument is true value.
3rd argument is false value.

if first is true, 2nd will be returned
otherwise 3rd will be returned.

Let us perform some conditional transformations

```
$ cat > file1
100,200
300,400
550,430
440,370
^d
hive> create table mytab(a int, b int)
hive> Insert overwrite table mytab
      row format delimited fields terminated by ',';
hive> load data local inpath 'file1' into table mytab;
hive> alter table mytab (big int);
hive> insert overwrite table mytab
      select a, b, If(a>b, a, b) from mytab;
hive> select * from mytab;
```

big

100	200	200
300	400	400
550	430	550
440	370	440

Nested If functions

```
$ cat > file2
100,200,300
200,400,50
300,120,270
^d
```

Among 3 colums which is biggest value.

```
hive> create table urtab(a int, b int, c int)
 row format delimited fields terminated by ',';
hive> load data local inpath 'file2' into table urtab;
```

```
hive> select * from urtab;
  -   -   -
  -   -   -

hive> Alter table urtab add columns (big int);

hive> Insert overwrite table urtab
        select a, b, c, If(a>b, If(a>c, a, c),
                           If(b>c, b, c))
                    from urtab;

hive> select * from urtab;
                          → big
100   200   300   300
200   400   50    400
300   120   270   300
```

Cleaning Null values

```
$ cat > file3
100,,200
200,,
,,300
 ,300,
10,20,
 ,30,40
100,50,25
^d

hive> create table Tab1(a int, b int, c int)
      row format delimited fields terminated by ',';

hive> load data local inpath 'file3' into table Tab1;

hive> select * from Tab1;
100   NULL   200
200   NULL   NULL
NULL  NULL   300
NULL  300    NULL
10    20     NULL
NULL  30     40
100   50     25
```

```
hive> select a+b+c from Tab1 ;
o/p →
NULL
NULL
NULL
NULL
NULL
NULL
175
```

because

100 + 200 = 300
NULL + 20 = NULL
200 - NULL = NULL

} In Arithmetic operation if Null values are existed final Result will be NULL

So bebfle Arithmetic operations we need to clean NULLs

I want to make NULLs as Zeros.

```
hive> Insert overwrite table Tab1
      select If(a is null, 0, a),
             If(b is null, 0, b),
             If(c is null, 0, c) from Tab1
```

```
hive> select * from Tab1;
100   0    200
200   0    0
0     0    300
0     300  0
10    20   0
0     30   40
100   50   25
```

```
hive> select a+b+c from Tab1;
300
200
300
300
30
70
175
```

Null values can not be compared with '=' and '!=' operators.

to be compared with "a is null" and its negation is "a is not null".

ex:-

hive> select * from emp where sal is null;

hive> select * from emp where sal != null;
↳ Invalid

hive> select * from emp where sal is not null;
↳ valid

Lets do some more transformations.

```
hive> create table newEmp (ecode int, name string,
        sal int, ~~dno~~ sex string, dno int)
        row format delimited fields terminated by ',';
hive> load data local inpath 'emp' into table newemp;
hive> select * from ~~emp~~ newemp;
```

101	AA	20000	M	11
102	BB	30000	F	12
⋮				

In Sex column,

'M' to be transformed as 'Male'

'F' to be tranformed as 'Female'

→

In dno column,

(dno) department numbers to be transformed as department names.

dno	→	dname
11	-as→	Marketing
12	→	HR
13	→	Finance
remaining	→	others

Sal column,

sal	→	grade
sal < 30000	→	D

sal >= 30000 & sal < 50000 → C

sal >= 50000 & sal < 70000 → B

sal >= 70000 → A

```
hive> create table etab (ecode int, name string,
        sal int, sex string, dname string,
          grade string);
hive> Insert overwrite table etab
        select ecode, name, sal,
            If (sex='F', 'Female', 'Male'),
            If (dno=11, 'Marketing',
                If (dno=12, 'HR',
                    If (dno=13, 'Finance', 'others'))),
            If (sal>=70000, 'A',
                If (sal>=50000, 'B',
                    If (sal>=30000, 'C', 'D')))
        from newEmp;
```

hive> select * from etab;

101	AA	20000	Male	Marketing	D
102	BB	30000	Female	HR	D
103	CCC	40000	Male	Finance	C
104	DD	70000	Female	Finance	A
105	FED	650000	Male	HR	B

⋮

COALESCE function.

→ it returns first not null value from the list of values.

→ if all values are nulls it returns null.

Example

$ cat > probiles

Ravi, 912345, 912346, 912347

Rani, , , 928216

Mani, , 937218, 937219

^d

hive> create table probiletab (name string, mobile int, residence int, offic int)
row format delimited fields terminated by ',';

hive> load data local inpath 'probiles' into table probiletab;

hive> select * from probiletab;

Ravi	912345	912346	912347
Rani	null	null	928216
Mani	null	937218	937219

hive> select name, coalesce(mobile, residence, offic)
from probiletab;

o/p →

Ravi	912345
Rani	928216
Mani	937218

Case statement in hive

We can also perform conditional transformations using case statement.

```
hive> select dno, case dno
            when 11 then 'Marketing'
            when 12 then 'HR'
            when 13 then 'Finance'
            else 'others'
            end
      from emp;
```

```
hive> select name, sal, case
            when sal>=70000 then 'A'
            when sal>=50000 then 'B'
            when sal>=30000 then 'C'
            else 'D'
            end
      from emp;
```

---

hive> use mydb;
hive> select * from raw; → only one column, line : string

```
xxxxx2010xxx23xx
xxxxx2010xxx-20xx
xxxxx2010xxx27xx
:
```

to extract year and temperature [Look at page hive-10]
In previous example we applied Union All.

```
select * from (
    Q1 ← { select substr(line,6,4), substr(line,13,2)
           from raw
           where substr(line,13,1) != '-'
         union All
    Q2 ← { select substr(line,6,4), substr(line,13,3)
           from raw
           where substr(line,13,1) = '-' ) tab ;
```

we merged outputs of two Queries Q1 and Q2.
Suppose raw table has 1Cr rows,
Q1 has to read 1Cr rows
and Q2 has to read 1Cr, total reading is 2Cr rows.
in just one table scan, we can read and extract year and temperature by using Conditional Transformation

~~hive> create table New as~~
~~select subs~~

→

```
hive> Create table New(y int, t int)
      row format delimited fields terminated by ';';

hive> Insert overwrite table New
      select substr(line, 6, 4),
      if (substr(line, 13, 1) != '-',
          substr(line, 13, 2),
          substr(line, 13, 3)) from raw;

hive> select * from New;
```

```
2010    23
2010    -20
2010    24
  :
```

If 13th character is '-' (hyphen)
it extracts positive temperature
otherwise it extracts negative temperature.

But substr() returns String value.

But this string contains pure numbers,
so it can be implicitly casted into int variables.
In our table y and t are int datatypes.

Aggregated functions in hive

SUM() → total of column
AVG() → average of column
MAX() → maximum value of column
MIN() → Minimum value of column
COUNT() → number of rows.

```
hive> select sum(sal) from emp;
hive> select sum(sal) from emp where sex='F';
hive> select avg(sal) from emp;
hive> select max(sal) from emp;
hive> select min(sal) from emp;
hive> select count(*) from emp;
hive> select count(sal) from emp;
```

↳ it will not count nulls of sal column.

How to make sure whether a column has null values or not ?

```
hive> select count(*) - count(sal) from emp;
```

if result is 0 → there are no nulls.
if result > 0 → there are nulls.

(0)

```
hive> select count(*) from emp where sal is null;
```

if result = 0 → no nulls
if result > 0 → nulls

Grouping Aggregations

Group by clause.

→ to perform aggregations seperately for each data group.

Single grouping with Single Aggregations

```
hive> select sex, sum(sal) from emp
        group by sex;
```

Sample outputs:

F	270000
M	300000

```
hive> select dno, avg(sal) from emp
        group by dno;
```

Single grouping with Multiple Aggregations

```
hive> select sex, sum(sal), avg(sal),
        max(sal), min(sal), count(*)
        from emp group by dno;
```

Multiple grouping Columns with Single Aggregation:

```
hive> select dno, sex, sum(sal) from emp
        group by dno, sex;
```

Sample o/p →

11	F	150000
11	M	200000
12	F	—
12	M	—
	⋮	

dno → Primary group; sex → Sub group

Hive 18

Multiple grouping columns with multiple Aggregations

```
hive> select dno, sex, sum(sal), avg(sal) from emp
        group by dno, sex;
```

having clause

to filter group items.

only applicable with 'Group by' clause.

```
hive> select dno, Sum(sal) from emp
        group by dno
          having dno in (11, 15);
```

11	?
15	?

Combination of where and having :—

```
hive> select dno, Sum(sal) from emp
        where sex='F' and sal>=40000
        group by dno
          having dno in (11, 15);
```

In this example, sex and sal are not grouping columns so you can not use them in having clause.

by dno you can use in where clause.

Its always better to filter grouping columns in HAVING clause for better performance.

Aggregated functions can be used in HAVING Clause:

```
hive> select dno, count(*) from emp
      group by dno
      having sum(sal) >= 100000;
```

→ valid

```
hive> select dno, count(*) from emp
      where sum(sal) >= 100000
      group by dno;
```

→ invalid.


```
hive> select dno, sum(sal) from emp
      where dno in (12,16)
      group by dno;
```

Row Filter happens at Mapper level

```
hive> select dno, sum(sal) from emp
      group by dno
      having dno in (12,16);
```

→ Filter happens at Reducer level.

Combination of ORDER BY and Group By

```
hive> select dno, sum(sal) as tot
        from emp
          group by dno
            order by tot desc
              limit 3;
```

Sample o/p →

13	370000
18	250000
11	150000

Top 5 pages from weblog table based on User visits

```
hive> select page, count(*) as cnt
        from weblog
          group by page
            order by cnt desc
              limit 5;
```

Eliminating Duplicates

- distinct() function is used to eliminate duplicates.

Tab

x
10
20
30
20
20
10

```
hive> select distinct(x) from Tab;
```

o/p →

10
20
30

How to make sure whether a column has duplicates or not?

```
hive> select count(*) - count(distinct(ecode))
        from emp;
```

If Result = 0 → no duplicate values.
if Result > 0 → duplicates existed.

How to find duplicate values in a column

```
hive> select ecode, count(*) as cnt
        from emp
        group by ecode
        having cnt > 1 ;
```

Table → samp

Columns →	id (int)	name (string)
	101	A
	102	B
	101	A
	102	B
	102	B
	103	C
	103	D

How to eliminate duplicate rows based on entire row match (If all columns are equal then row treated as duplicate)?

```
hive> select distinct(id), name from samp;
```

O/P →
101 A
102 B
103 C
103 D
} → All columns not matched

But to eliminate duplicates based on same column, we will have to use UDF's. [In UDF part we will see]

We can also eliminate duplicate rows by using group by clause

emp

ecode	ename	sal	sex	dno
101	A	20000	M	11
102	B	30000	F	12
101	A	20000	M	11
102	B	30000	F	12
103	C	50000	M	13

```
hive> select ecode, ename, sal, sex, dno
        from emp
        group by ecode, ename, sal, sex, dno;
```

o/p →

101	A	20000	M	11
102	B	30000	F	12
103	C	50000	M	13

In both techniques such as distinct() or group by, what if, if your table has 500 columns?
We can not list out all columns.
then follow the following simple technique

```
hive> create table dummy (line string);
hive> load data inpath '/user/hive/warehouse/emp'
        into table dummy;
hive> insert overwrite table dummy
        select distinct(line) from dummy;
```

→

hive> load data inpath '/user/hive/warehouse/dummy'
into table emp;
hive> drop table dummy;
hive> select * from emp;

101 A 20000 M 11
102 B 30000 F 12
103 C 50000 M 13

By this technique, If you have 100 columns not a problem and 1000 columns not a problem.

Merging Tables

UNION ALL

In Sql (RDBMS) we have two Unions

(i) Union
(ii) Union All

Union will not allow duplicate rows
Union All will allow duplicate rows.

hql has only Union All, which allows duplicates. But if you dont want duplicates, first merge tables, later apply distinct or group by then eliminate duplicate rows.

→ In hql, Union should be a sub query,
→ Sub query should have an alias

Table → emp1

ecode	name	sal	sex	dno
101	A	20000	M	11
102	B	30000	F	12
⋮				

→ 100 rows

Table → emp2

ecode	name	sal	sex	dno
201	AA	50000	F	13
202	BB	60000	M	12
⋮				

→ 200 rows

Task:- merge all rows of emp1 and emp2 into a table emp;

In this case tables emp1, emp2, emp schema is same → structure.

```
hive> Insert overwrite table emp
        select * from (
          select * from emp1
            union All
          select * from emp2 ) e ;

hive> select * from emp ;
```

= } → 100 + 200 = 300 rows.

What if, if tables have different structures?

emp1 → ecode, name, sal, sex, dno

emp3 → ecode, name, dno, sal, sex

```
hive> Create table stb like emp;
hive> Insert overwrite table stb
        select * from (
            select ecode, name, sal, sex, dno from emp1
              Union all
            select ecode, name, sal, sex, dno from emp2
        ) e;
```

-- we should not use '*' in select statement, if schema is different.

In above two union examples, number of columns in both tables is same.

How to merge if number of columns are different?

Tab1 → ecode, name, sal, sex, dno, designation
Tab2 → ecode, name, sal, dno, city

Tab1 does not have city column
Tab2 does not have sex and designation

```
hive> create table TabX(ecode int,
        name string, sal int, dno int,
        sex string, city string, designation string);
```

→

hive-22

```
hive> Insert overwrite table Tabx
        select * from (
      select ecode, name, sal, dno, sex,
          ' ' as city, designation
          from Tab1
          union all
      select ecode, name, sal, dno, ' ' as sex,
          city, ' ' as designation
          from Tab2 ) tab;
```

```
hive> select * from Tabx;
```

Sample o/p

ecode	name	sal	dno	sex	city	desig
101	AA	20000	11	M		SE
102	BB	30000	12	F		JSE
201	CC	40000	11		Hyd	
202	DD	50000	12		Del	

You can perform grouped aggregations over multiple tables:

model1

```
hive> select dno, sum(sal) from (
        select dno, sal from emp1
          union all
        select dno, sal from emp2 ) e
          group by dno;
```

you can do last query, as follows.

Model2

```
hive> select dno, sum(tot) as totsal from (
        select dno, sum(sal) as tot from emp1
          group by dno
        union all
        select dno, sum(sal) as tot from emp2
          group by dno ) e
          group by dno;
```

→ Model1 is best performance.

How to append rows of one table to another table?

ex:- I want to append rows of table Taby to table tabx.

tabx ~~taby~~ → prid, amt (with delimiter comma)

~~taby~~ tabx

prid	amt
p1	20000
p2	30000
p3	25000
⋮	

5 lakh rows

As already you know how to create table with Comma delimiter

taby ~~tabx~~ → prid, amt (with comma delimiter)

~~tabx~~ taby

prid	amt
p1	25000
p3	27000
⋮	

200 rows.

We need to append all 200 rows of taby to tabx.

If you use "~~Enter~~ "Insert overwrite" target table's data will be overwritten

So we use Union All (Merge technique)

```
hive> Insert overwrite table TabX
        select * from (
          select * from TabX      -> 5Lakh
            union all             -> 200
          select * from Taby ) tab;
```

But this approach is bad, because, just to append 200 rows of taby, hive is reading 5lakh and 200 rows. What if, if tabx has 5cr rows?

So follow the following technique,

→ copy the backend files of Taby to backend hdfs directory of TabX.

```
$ hadoop fs -cp /user/hive/warehouse/taby
                /user/hive/warehouse/tabx
```

now tabx contains all data of previous tabx and taby.

by this technique, no need to read rows of any table while appending.

but approach is wrong if taby has a different delimiter.

Tabx → prid, amt (with Comma delimiter)

Tabz → prid, amt (with tab delimiter)
↳ in backend file.

hive> Create table Dummy like Tabx;

hive> Insert overwrite table Dummy
select * from Tabz;

now, backend file of Dummy table has Comma delimiter.

$ hadoop fs -cp /user/hive/warehouse/Dummy/000000_0
/user/hive/warehouse/Tabx ↵

if tabx has a file with name 000000_0

then,

$ hadoop fs -cp /user/hive/warehouse/Dummy/000000_0
/user/hive/warehouse/tabx/newfile ↵

hive> drop table Dummy;

If you want to append only p2, p4 transactions of Taby to tabx. ?

hive> create table dummy like taby;
hive> insert overwrite table dummy
select * from taby where prid in ('p2', 'p4');

$ hadoop fs -cp /user/hive/warehouse/dummy/000000_0
/user/hive/warehouse/tabx/filey

Now table dummy contains only p2, p4 transactions of taby, those rows appended to tabx, with out reading rows of tabx.

Joins Using Hive

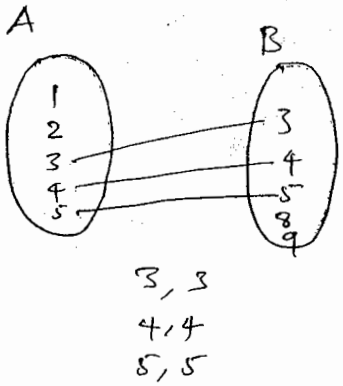
Joins are used to collect data from two or more tables, based on some joining column.

Joins are two types basically.

(i) Inner Joins
(ii) Outer Joins

Inner Join

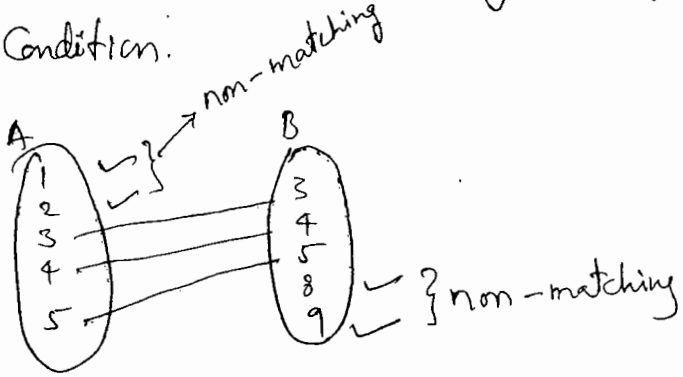
Only matching rows with join Condition.



3, 3
4, 4
5, 5

Outer Joins

matching rows and non matching rows with join Condition.



Outer Joins are 3 types.

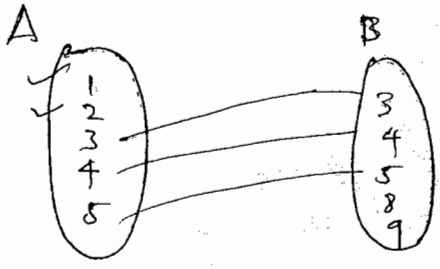
→ Left outer Join

→ Right outer Join

→ Full Outer Join

Left Outer Join :-

→ All matching rows and non matchings of left side table



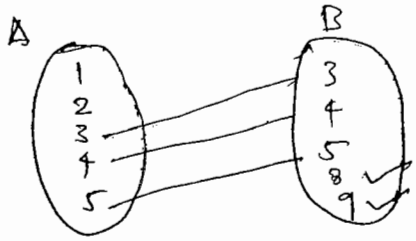
1, Null
2, Null
} non matchings from Left

+

3, 3
4, 4
5, 5
} matchings

Right outer Join :-

→ All matching rows and non-matchings of Right side table.



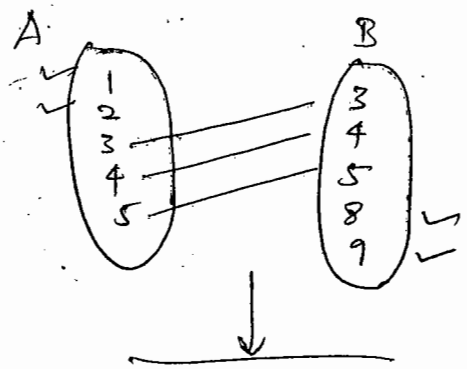
3, 3
4, 4
5, 5
} → matchings

null, 8
null, 9
} → non-Matchings from Right.

BY: SREERAM (bharat.sreeram@gmail.com)

Full outer Join

→ All matching rows and non matchings of left side table, non matchings of Right side table



3, 3
4, 4
5, 5
} → matchings

1, Null
2, Null
} → Non matchings of Left side

Null, 8
Null, 9
} → Non matchings of Right side.

Emp

ecode	name	sal	sex	dno
101	Amar	20000	M	11
102	Amala	30000	F	12
103	Ankit	40000	M	11
104	Ankita	50000	F	11
105	Anush	60000	M	13
106	Anusha	70000	F	12
107	Akhil	80000	M	14
108	Akhila	90000	F	15

Dept

dno	dname	dloc
11	Marketing	Hyd
12	HR	Del
13	Finance	Hyd
16	Accounts	Del
17	CRM	Hyd

ecode	name	sal	sex	emp.dno	dept.dno	dname	dloc	
101	Amar	20000	M	11	11	Marketing	Hyd	①
102	Amala	30000	F	12	12	HR	Del	①
103	Ankit	40000	M	11	11	Marketing	Hyd	①
104	Ankita	50000	F	11	11	Marketing	Hyd	①
105	Anush	60000	M	13	13	Finance	Hyd	①
106	Anusha	70000	F	12	12	HR	Del	①
107	Akhil	80000	M	14	NULL	NULL	NULL	②
108	Akhila	90000	F	15	NULL	NULL	NULL	②
NULL	NULL	NULL	NULL	NULL	16	Accounts	Del	③
NULL	NULL	NULL	NULL	NULL	17	CRM	Hyd	③

① → Inner Join

① + ② → Left outer join

① + ③ → Right outer join

① + ② + ③ → Full outer join

Inner Join

```
hive> select ecode, name, sal, sex, dname, dloc
      from emp e join dept d
      on (e.dno = d.dno);
```

Left outer Join

```
hive> select ecode, name, sal, sex, dname, dloc
      from emp e join dept d
      on (
```

```
hive> select ecode, name, sal, sex, dname, dloc
      from emp e Left outer join dept d
      on (e.dno = d.dno);
```

Right outer join

```
hive> select ecode, name, sal, sex, dname, dloc
      from emp e Right outer join dept d
      on (e.dno = d.dno);
```

Full outer Join

```
hive> select ecode, ename, sal, sex, dname, dloc
      from emp e Full outer join dept d
      on (e.dno = d.dno);
```

Task,

For each city, total salary is required

In this task city is in dept table and sal is in emp table. So joins required.

```
hive> select dloc, sum(sal)
      from emp e join dept d
      on (e.dno=d.dno)
      group by dloc;
```

o/p →

Del	100000
Hyd	170000

But everytime while collecting data from multiple tables, if you apply joins, it will take lots of process time.

So denormalize the tables into a single table.

Denormalization

```
hive> create table EmpDept (ecode int, name string,
      sal int, sex string, dno1 int, dno2 int,
      dname string, dloc string)
      row format delimited fields terminated by ',';
```

-- Let's apply Full outer join, so no data is missed.

```
hive> Insert overWrite table EmpDept
      select ecode, name, sal, sex, e.dno, d.dno,
       dname, dloc
       from emp e Full outer join dept d
        on (e.dno = d.dno);
```

Now, all rows of emp, dept available in EmpDept.

Now with out applying joins everytime, you can directly query on Empdept.

ex:-

```
hive> select dloc, sum(sal) from empdept
        group by dloc;
```

o/p →

NULL	170000
del	100000
hyd	170000

We loaded "Full outer join" content into empdept which contains matching tuples (rows), and non matching rows based on join condition (e.dno = d.dno)

To get matching Rows

```
hive> select * from empdept where dno1 = dno2;
```

To get matching Rows and non matching rows of left

```
hive> select * from empdept where dno1 is not null;
```

To get matching rows and non-matchings of right

hive> select * from empdept where dno2 is not null;

To get only non-matchings of left side

hive> select * from empdept where dno2 is null;

To get only non-matching rows of Right side :-

hive> select * from empdept where dno1 is null;

To get only non matchings of both left and right :-

hive> select * from empdet where dno1 is null or
dno2 is null;

We do some transformations

treat 'dno' as projectid.

if you observe emp, dept tables, employees of 11, 12, 13 dnos engaged into projects, such as Marketing, HR, Finance.

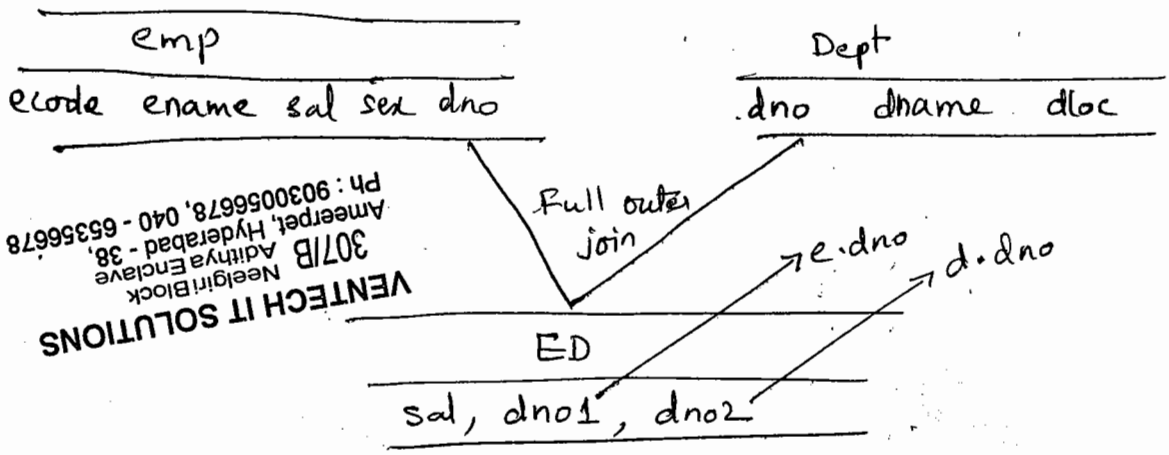
14, 15 employees not engaged into any project.

Treat 14,15 employees as bench Team.

16, 17 dnos have no employees. Treat them as bench projects.

Now I need following Report

Status	TotalSalary
Working	270000
BenchTeam	170000
BenchProject	0




```
hive> create table ED(sal int, dno1 int, dno2 int);

hive> Insert overwrite table ED
        select sal, e.dno, d.dno from
          emp e full outer join dept d
          on (e.dno = d.dno);

hive> select * from ED;
```

20000	11	11
40000	11	11
50000	11	11
30000	12	12
70000	12	12
60000	13	13
80000	14	NULL
90000	15	NULL
NULL	NULL	16
NULL	NULL	17

```
hive> create table estat(stat String, sal int);

hive> Insert overwrite table estat
        select If(dno1=dno2,'Working',
          if(dno2 is null,'BenchTeam','BenchProject')),
          sal from ED;
```

```
hive> select * from estat;
Working   20000
Working   40000
Working   50000
Working   30000
Working   70000
Working   60000
BenchTeam 80000
BenchTeam 90000
BenchProject NULL
BenchProject NULL
```



```
hive> create table result(stat string, totsal int)
      row format delimited fields terminated by ',';
hive> insert overwrite table result
      select stat, sum(sal) from estat
      group by stat;
hive> insert overwrite table result
      select stat, if(totsal is null, 0, totsal) from
      result;
hive> select * from result;
```

o/p

```
BenchProject   0
BenchTeam      170000
Working        270000
```

Hive Joins dont support inequality conditional operators in Join Condition.

Inequality Conditions

!=
>
<
>=
<=

LEFT SEMI JOIN

table1

name
amar
amala
ankit
ankita
anush
anusha

table2

name
amar
ankit
ankit
amar
amar

I want to get names of table1, that only appear in table2

```
hive> select l.name from
        table1 l Left semi join table2 r
          on (l.name = r.name);
```

o/p →
amar
ankit

o/p of inner join is as follows

```
hive> select l.name fran table1 l join table2 r
        on (l.name = r.name);
```

o/p →
amar
amar
amar
ankit
ankit

JOINS with SUB QUERIES
→ NESTED SELECT statements

Emp

name	salary
A	10000
B	20000
C	30000
D	40000
E	70000
F	90000
G	70000
H	70000
I	90000
J	80000
K	80000

To Get Top 3 salaried employees

generally we apply following statement,

```
hive> select * from Emp
        order by salary desc
          limit 3;
```

o/p →

F	90000
I	90000
J	80000

But in Emp table,
top salary is 90000, taken by 2 people.
80000 (2nd max) taken by 2 people (J, K)
70000 (3rd max) taken by 3 people (E, G, H)

```
hive> select l.* from emp l join (
        select distinct(salary) from emp
          order by salary desc limit 3) r
        on (l.salary = r.salary);
```

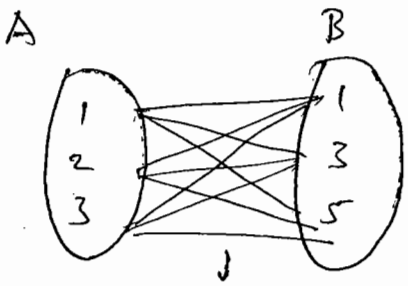
o/p →

E	70000	Top 3rd
G	70000	
H	70000	
J	80000	Top 2nd
K	80000	
F	90000	Top 1
I	90000	

Hive joins dont support inequality Conditions in ON clause we can use Cartisian product (cross join) to do inequality operations. Using where clause we can filter rows.

Cartisian product

Each row of Left side table join with each row of right side table.



J

1, 1
1, 3
1, 5
2, 1
2, 3
2, 5
3, 1
3, 3
3, 5

Task1 using Cartisian product

To Get 2nd Max salary

```
hive> select max(salary) from memp l
      join (select max(salary) as m from memp) r
      where l.salary < r.m ;
```

o/p → 80000

In above statement l.salary < r.m is inequality Condition.

Task2

How many employees are below Average salary and How many are above Average Salary?

~~hive> Alter table emp add columns~~

```
hive> Create table emp2 as
        select * from emp;
hive> alter table emp2 add columns (avg int);
hive> Insert overwrite table emp2
        select name, salary, mean from emp2 l
      join (select avg(salary) as mean from emp2) r;
```

o/p →

name	salary	Avg
A	10000	59090
B	20000	59090
C	30000	59090
⋮	⋮	
K	80000	59090

} now Avg(salary) is available for each row.

Hive

```
hive> alter table emp2 add columns (stat string);
hive> insert overwrite table emp2
        select name, salary, avg,
            if (salary >= avg, 'above', 'below')
        from emp2;
hive> select * from emp2;
```

name	salary	avg	stat
A	10000	59090	Below
⋮	⋮		
K	80000	59090	above

```
hive> select stat, count(*) from emp2
                group by stat;
```

o/p →

above	7
below	4

Test

Lets work with Sales Transactions

Amount

```
$cat Sales ↵      [this file has two fields date and amt]
01/13/2001, 20000
01/26/2001, 30000
    ⋮
12/07/2001, 50000
12/31/2001, 70000
    ⋮
10/25/2014, 200000
10/26/2014, 700000
    ⋮
12/29/2014, 1000000
12/31/2014, 200000
```

```
hive> create table sales (dt string, amt int)
      row format delimited fields terminated by ',';
hive> load data local inpath 'Sales' into table sales;
hive> select * from sales;
01/13/2001, 20000
```

this date is not in hive date format.

Lets transform MM/DD/YYYY format to YYYY-MM-DD format
↳ hive date format.

```
hive> create table sales2 (dt array<string>, amt int);
hive> insert overwrite table sales2
      select split(dt,'/'), amt from sales;
```

string, delimiter

split() returns String Array.

split('a,b,c',',') → ["a","b","c"]

split('01/03/2001','/') → ["01","03","2001"]

```
hive> create table sales3 like sales;
hive> Insert overwrite table sales3
      select concat(dt[2],'-',dt[0],'-',dt[1]),
             amt from sales2;
hive> select * from sales3;
2001-01-03      20000
.
.
.
2014-12-31      2000000
```

NOW date is in hive format,
you can apply all hive date functions
Such as month(), day(), year()

month('2001-01-03') → 1

day('2001-01-03') → 3

year('2001-01-03') → 2001

yearly Sales Report

```
hive> create table yearlyReport(y int, totSales int);
hive> Insert overwrite table yearlyReport
        select year(dt), sum(amt) from sales
          group by year(dt);
```

Monthly Sales Report for year 2014

```
hive> create table Report2014(m int, totSales int);
hive> Insert overwrite table Report2014
        select month(dt), sum(amt) from sales
          where year(dt)=2014
          group by month(dt);
```

Daily Sales Report (Day wise) for month December of 2014

```
hive> create table Report2014_December(d int, totSales int);
hive> insert overwrite table Report2014_December
        select day(dt), sum(amt) from sales
          where year(dt)=2014 and month(dt)=12
          group by day(dt);
```

Quarterly Sales Report for year 2014

Quarter 1 → 1, 2, 3 months
Quarter 2 → 4, 5, 6 "
Quarter 3 → 7, 8, 9 "
Quarter 4 → 10, 11, 12 "

```
hive> alter table sales3 add columns (quarter int);
hive> Insert overwrite table sales3
        select dt, amt, case
                when month(dt)<4 then 1
               when month(dt)<7 then 2
               when month(dt)<10 then 3
              else 4
              end from sales3;
-- for all rows quarter number is generated.

hive> create table Quarterly_Report2014(quarter int,
                totamt int);
hive> Insert overwrite table Quarterly_Report2014
        select quarter, sum(amt) from sales3
              group by quarter;
```

Half yearly Sales Report for each year. (Multi grouping)

```
hive> alter table sales3 add columns (h int);
hive> insert overwrite table sales3
        select dt, amt, quarter,
                if(quarter < 3, 1, 2) from sales3;
hive> create table Halfyearly_Report(y int, h int,
                totsales int);
hive> Insert overwrite table Halfyearly_Report
        select year(dt), h, sum(amt)
              from sales3
             group by year(dt), h;
```

Lets come back to Cartisian product

from previous sales Repots, lets take Quarterly_Report2014

```
hive> select * from Quarterly_Report2014;
```

Sample o/p

quarter	totamt
1	500000
2	750000
3	600000
4	900000

Now the task is,

for each quarter, when Compared with its previous quarter sales, how much percentage of sales increased or decreased.

ex:- 2nd quarter Sales increased for 50% when Compared with 1st quarter Sales.

Now each quarter's sales volume to be Compared with its previous Row's sales volume.

its not possible.

so we can take help of "Cartisian product"

```
hive> create table cart1 (q1 int, tot1 int,
                          q2 int, tot2 int);

hive> Insert overwrite table cart1
      select l.quarter, l.totamt, r.quarter, r.totamt
        from quarterly_report2014 l
             join
             quarterly_report2014 r;
```

```
hive> select * from cart1;
```

q1	quarter1	totamt1	tot1	quarter2	q2	totamt2	tot2
	1	500000		1		500000	
	1	500000		2		750000	
	1	500000		3		600000	
	1	500000		4		900000	
		⋮					

in this way each quarter joind with each other quarter.

we need only 2 — 1
3 — 2
4 — 3

so, at the time of cartisoning apply row filter "where quarter1 - quarter2 = 1"

```
hive> insert overwrite table cart1
      select l.quarter1, totamt1,
             quarter2, totamt2
      from
```

```
hive> insert overwrite table cart1
      select l.quarter, l.totamt, r.quarter,
             r.totamt from
      quarterly_report2014 l join quarterly_report2014 r
      where l.quarter - r.quarter = 1;
```

```
hive> select * from cart1;
```

O/P →

q1	quarter1	tot1 totamt1	q2	quarter2	tot2 totamt2
	2	750000		1	500000
	3	600000		2	750000
	4	900000		3	600000

```
hive> alter table cart1 add columns
        (percent_growth float);

hive> insert overwrite table cart1
      select q1, tot1, q2, tot2,
        ((tot1-tot2)*100)/tot2 from cart1;

hive> select * from cart1;
```

q1	tot1	q2	tot2	percentage_growth
2	750000	1	500000	50.0
3	600000	2	750000	-20.0
4	900000	3	600000	50.0

$$\text{Percentage growth formula} = \left[\frac{\text{Present value} - \text{Previous value}}{\text{previous value}} \times 100\right]$$

Hive Partitioned tables

up to now we know,
Hive tables are two types
(i) Inner tables (Managed-Tables)
(ii) External tables.

Another classification of tables

(i) Partitioned Tables.
(ii) Non-partitioned Tables.

In case of Inner & External tables,
by default each table is Inner (Managed).

In partitioned/non-partitioned, by default every table is "Non-partitioned"

What is a partition?

Partition is a sub directory of Table's directory.

Advantage of Partitioned Table

FASTER ACCESS,

When you fetch data from non-partitioned tables, hive has to read all rows of all files of table's backend directory.

This is good, when you want all rows of a table.

See the following example:

Table 'sales' has 10 crore records (rows)
[may be single file/multiple files in table's backend directory of HDFS].
'sales' tables has transactions history from 2001-01-01 to till date,
When you want to fetch only a perticular date (ex:- 2015-07-19 trasactions) you will give following statement.

```
hive> select * from Sales
      where dt ='2015-07-19';
```

Suppose your output is 100 Records,
Now observe the problem → just to fetch 100 Rows of that date, hive is reading All rows of the table (10 crore rows).
Which cause high latency, is expensive in terms of time and CPU Resources.
If you are able to keep each day trasactions in to a seperate partition,
When you request that date, hive will point that partition and reads all rows of that partition, with out reading of all rows of the table. It saves a lots of Reading time.

Examples on Partitioned Tables

non-partitioned

```
hive> use mydb;
hive> create table emp (ecode int, name string,
        sal int, sex string, dno int)
      row format delimited fields terminated by ',';

hive> load data local inpath '/home/ventech/emp.txt'
      into table emp;
```

Assume, emp table as 100 males and 20 females
total 120 rows.

```
hive> select * from emp where sex = 'f';
```

-- now hive has to read all 120 rows, after reading each row it validates the condition (sex='f'), if true then it writes row.

Now I want to keep all females in to 1 partition, and all males into another partition.

creating partitioned table.

```
hive> create table epart (ecode int, name string,
        sal int, sex string, dno int)
      partitioned by (sex string);
```

This query will be failed. because column 'sex' repeated in general columns and partioned columns. Both should have unique name.

```
hive> Create table epart (ecode int, name string,
      sal int, sex string, dno int)
      partitioned by (s string)
      row format delimited fields terminated by ';';
```

→ s for sex (different name)

```
hive> describe epart;
ecode   int
name    string
sal     int
~~string~~
sex     string
dno     int
```

} Physical columns (backend file contains only 5 fields)

```
s       String
```

} → logical column (partitioned column)

Now table has 5 physical columns and 1 logical column total 6 columns.

In HDFS,

/user/hive/warehouse/mydb.db/epart

you will find only table directory 'epart'. No subdirectories for partitions.

NOTE:- Partitions are created at the time of loading data into partitions, not at the time of creating tables.

→ We can load data from file to partitions and table to partitions.

Initially we work with loading from tables (non-partitioned) to partitioned tables.

LOADING DATA FROM NON-PARTITIONED TABLE TO PARTITIONED TABLE

```
hive> insert overwrite table epart
        select * from emp where sex='F';
```

This query will be failed, because, epart is not a normal table. It is a partitioned table.

If you are inserting into partitioned table, you need to specify partitioned name.

Correct the statement as follows.

```
hive> insert overwrite table epart
        partition (S='F')
        select * from emp where sex='F';
```

epart → partitioned table

partition (S='F') → partition name.

emp → non-partitioned table

~~hive>~~ NOW you loaded all female rows into S=F partition.

do same thing for males.

```
hive> insert overwrite table epart
        partition (S='M')
        select * from emp where sex='M';
```

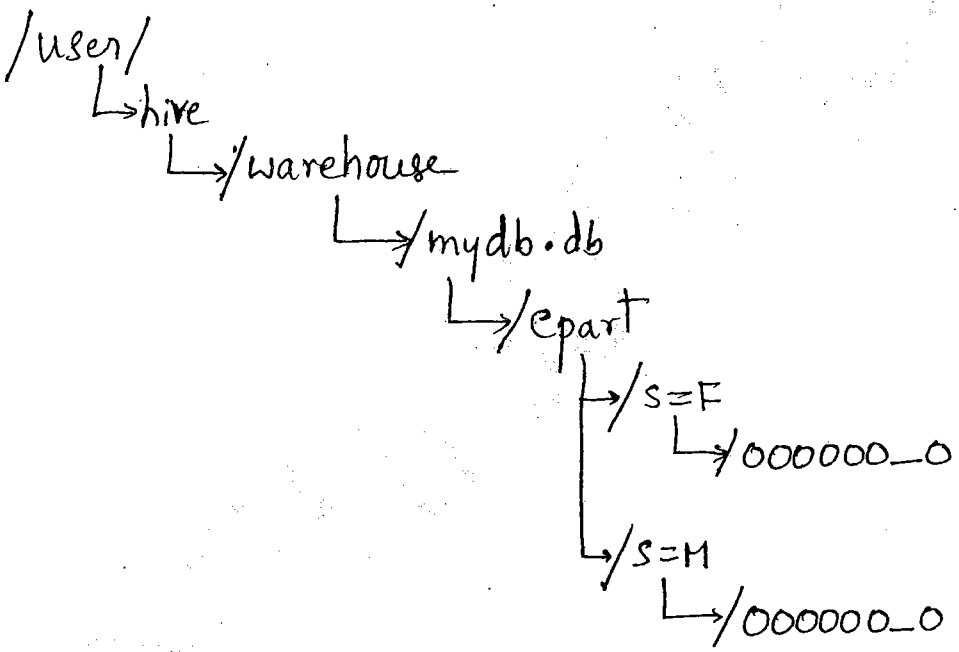
In above ~~two~~ statements, two Seperate statements we used one for S=F partition 2nd for S=M partition.

hql Supports Multi Loading (Multi insertions) in Single statement →

Multiple Insertion

```
hive> from emp
      Insert overwrite table epart partition (s='F')
        select * from where sex='F'
      Insert overwrite table epart partition (s='M')
        select * where sex='M';
```

In HDFS,



Observe above graph.

epart → is directory for table.

S=F → is subdirectory of epart

S=F/000000_0 file contains only female data

S=M → is subdirectory of epart

S=M/000000_0 file contains only male data.

Reading data from partitions

hive> select * from epart;
→ hive reads all partitions.

hive> select * from epart where sex='F';
→ hive reads all partitions, because 'sex' is not partition column. 's' is our partition column.
to take advantage of partitions, you should use partition column in where clause

hive> select * from epart where s='F';

now hive directly points
/user/hive/warehouse/mydb.db/epart/s=F partition
and read all rows of 000000_0 file, with out checking condition. ~~and reading all~~.
So for this query, no need of reading all rows of table. So lot reading time is saved.

Creating Partitioned tables using multiple columns:—

```
hive> create table mpart(ecode int, name string,
        sal int)
      partitioned by (dno int, sex string)
      row format delimited fields terminated by ',';
hive> describe mpart;
```

ecode int
name string } → physical columns
sal int
dno int
sex string } → logical columns for partitions.

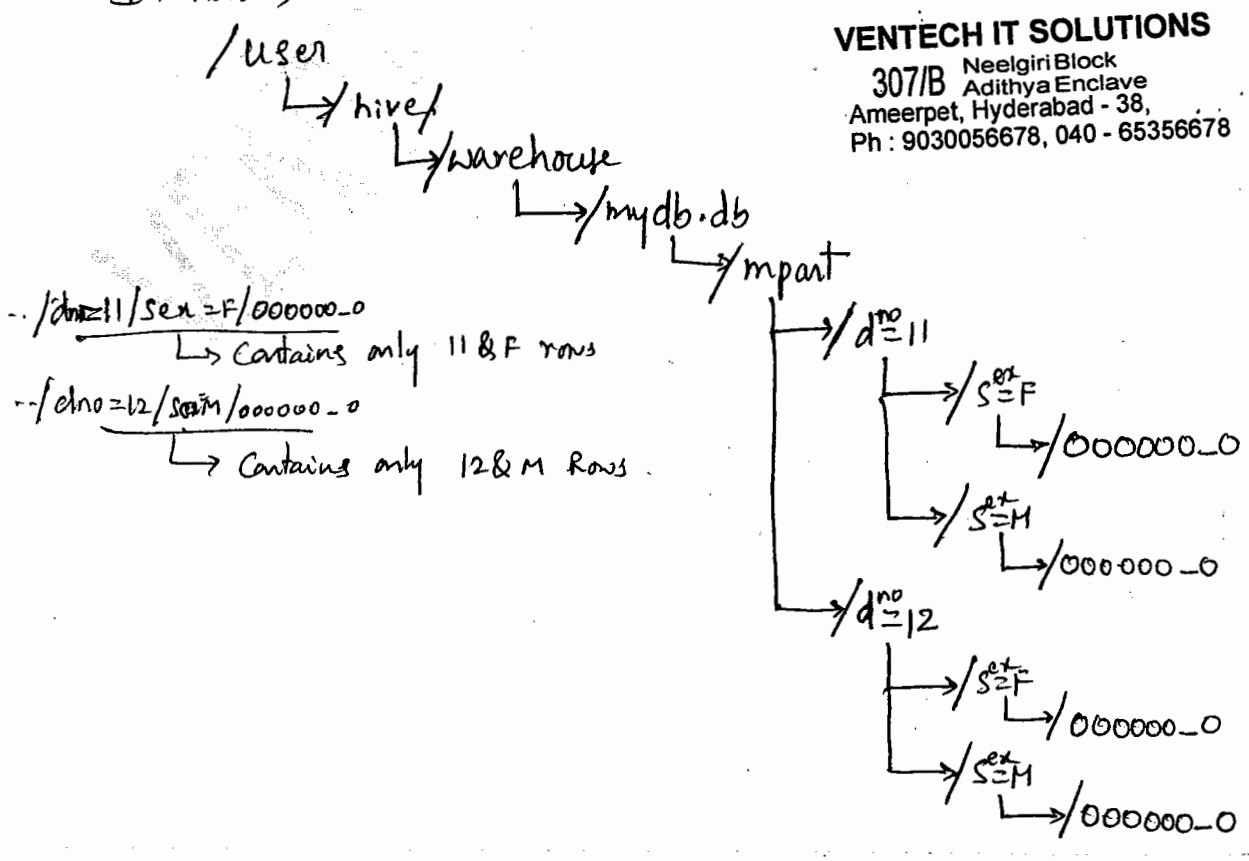
```
hive> from emp
      Insert overwrite table mpart partition(dno=11, sex='F')
          select ecode, name, sal where
                 dno=11 and sex='F'

      Insert overwrite table mpart partition(dno=11, sex='M')
          select ecode, name, sal, where
                 dno=11 and sex='M'

      Insert overwrite table mpart partition(dno=12, sex='F')
          select ecode, name, sal where
                 dno=12 and sex='F'

      Insert overwrite table mpart partition(dno=12 and
                                             sex='M')
          select ecode, name, sal where
                 dno=12 and sex='F';
```

In hdfs,



NOW, you created mpart table,
with Partitioned by (dno int, sex string)

here, dno is primary partition for mpart
and sex is sub partition for dno

-primary partition is a subdirectory of table's directory
and sub partition is a subdirectory of primary partition.
In this way you can go for multi level sub partitioning.

Reading data from multiple partitions

hive> select * from mpart;
→ hive reads all rows of all partitions

hive> select * from mpart where dno=12;
hive reads
/user/hive/warehouse/mydb.db/mpart/dno=12/sex=F/000000_0
/user/hive/warehouse/mydb.db/mpart/dno=12/sex=M/000000_0

hive> select * from mpart where dno=12 and sex='M';
hive reads only
/user/hive/warehouse/mydb.db/mpart/dno=12/sex=M/000000_0

hive> select * from mpart where sex='F';
hive reads
/user/hive/warehouse/mydb.db/mpart/dno=11/sex=F/000000_0
/user/hive/warehouse/mydb.db/mpart/dno=12/sex=F/000000_0

In previous example, mpart
we have loaded 11, M
11, F
12, F
12, M partitions.

2 departments (dno), 2 sex groups.

So 2 x 2 = 4 partitions.

So we used 4 Insert statements for loading.

If we have 100 departments (dno) in emp table, then we need to load into 100 x 2 = 200 partitions, which is very expensive process.

We need dynamic loading facility

Hive supports dynamic loading into partitions.

By default, dynamic loading feature is disabled in hive.

set the following option to enable.

hive> set hive.exec.dynamic.partition=true;

Loading data into partitions dynamically

hive> create table dpart like mpart;

hive> Insert overwrite table dpart partition (dno,sex)
select ecode, name, sal, dno, sex
from emp;

this query will be failed, because
primary partition can not be dynamic
except primary partition (dno), remaining partitions are dynamic.

In table dpart, dno can be static
and sex can be dynamic.

static dynamic

```
hive> from emp
    Insert overwrite table dpart partition (dno=11, sex)
      select ecode, name, sal, sex from emp where dno=11
    Insert overwrite table dpart partition (dno=12, sex)
      select ecode, name, sal, sex from emp where dno=12;
```

But, we need primary partition (dno) also as dynamic
then set one more following option.

~~hive> set~~

```
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

→ by default this option value is "strict".

Now we can load dynamically into primary and its sub partitions

```
hive> Insert overwrite table dpart partition (dno, sex)
        select ecode, name, sal, ~~from emp~~ dno, sex
          from emp;
```

So to load dynamically into partitions, we need to set two options

(i) set hive.exec.dynamic.partition=true;

(ii) set hive.exec.dynamic.partition.mode=nonstrict;

these options are temporary, once you quit from hive shell these will be set back to default values as FALSE and STRICT

Lets work on Sales table

```
hive> create table sales (dt string, amt int)
      row format delimited fields terminated by ',';
hive> load data local inpath '/user/ventech/sales'
      into table sales;
hive> select * from sales;
```

dt	amt
2001-01-01	250000
⋮	
2015-07-19	3000000

I want to create a partitioned table,
with year as primary partition
month is sub-partition of year
and day is sub-partition of month,

→almost 15 years data,
under each year 12 months
under each month, 30 (30/31/28) days
~~finds~~ on each day average 100 records (transactions)

```
hive> create table newsales (dt string, amt int)
      partitioned by (y int, m int, d int)
      row format delimited fields terminated by ',';
```

→

Now, loading dynamically into NewSales table from Sales table.

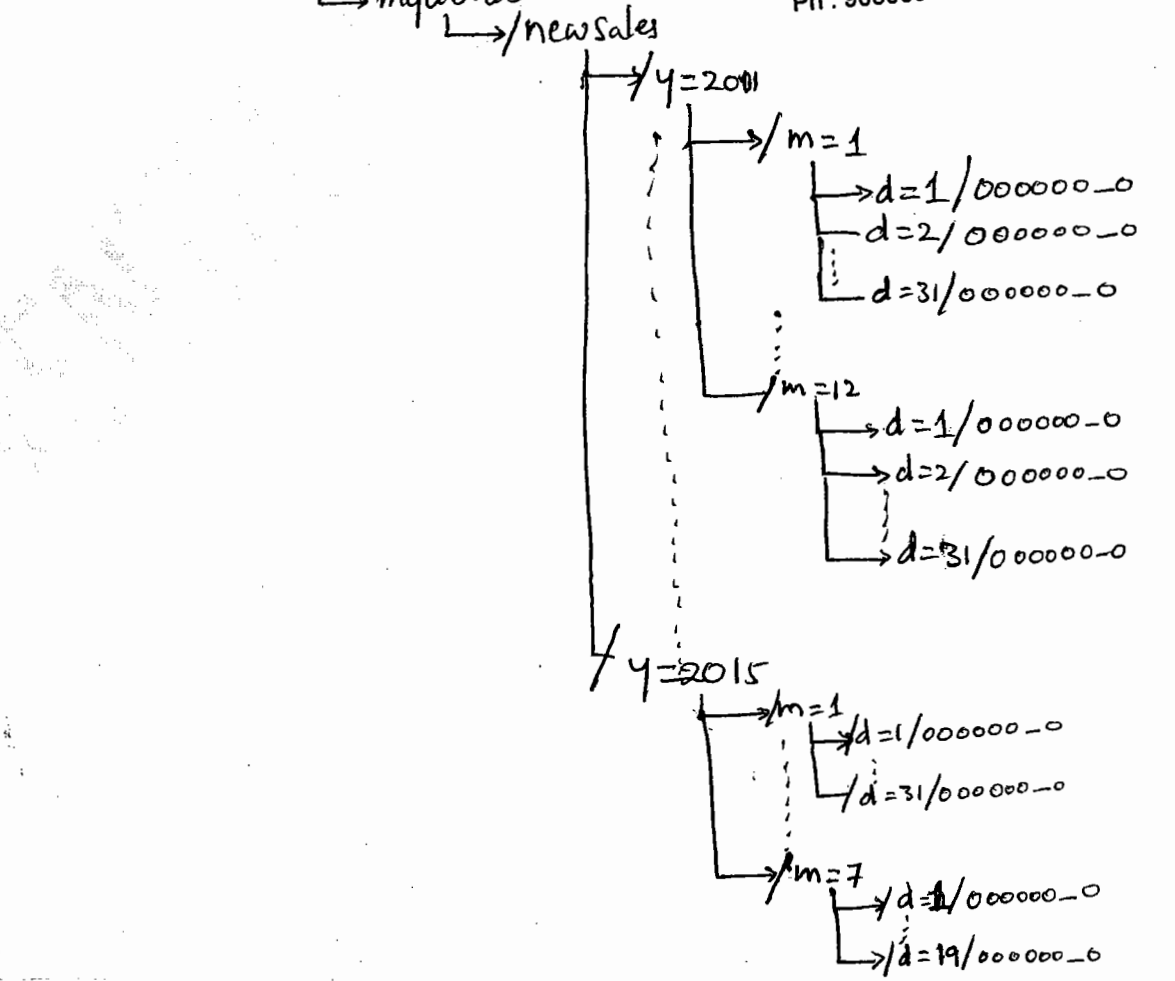
```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

→ to enable Dynamic loading

→ to make primary partition as dynamic.

```
hive> insert overwrite table newsales
        partition (y, m, d)
      select dt, amt, year(dt), month(dt), day(dt)
       from sales;
```

In HDFS,



hive> select * from newsales;

→ hive reads all partitions.

hive> select * from newsales where y=2014;

→ hive reads 365 partitions.

how?, 2014 has 12 Sub partians for month.

Each Month has 30/31/28 Sub partians for day.

hive> select * from newsales where m=1;

→ hive reads 15X31 partitions.

because m=1 partition available in each year (15)

& m=1 as 31 partitions for day.

hive> select * from newsales where y=2015 and m=6 and d=20;

→ hive reads only one partition.

Task

I want to fetch Sales transactions between 2003 may to 2010 july.

hive> select * from newsales

where (y=2003 and m>=5)

or

(y>2003 and y<2010)

or

(y=2010 and m<=7);

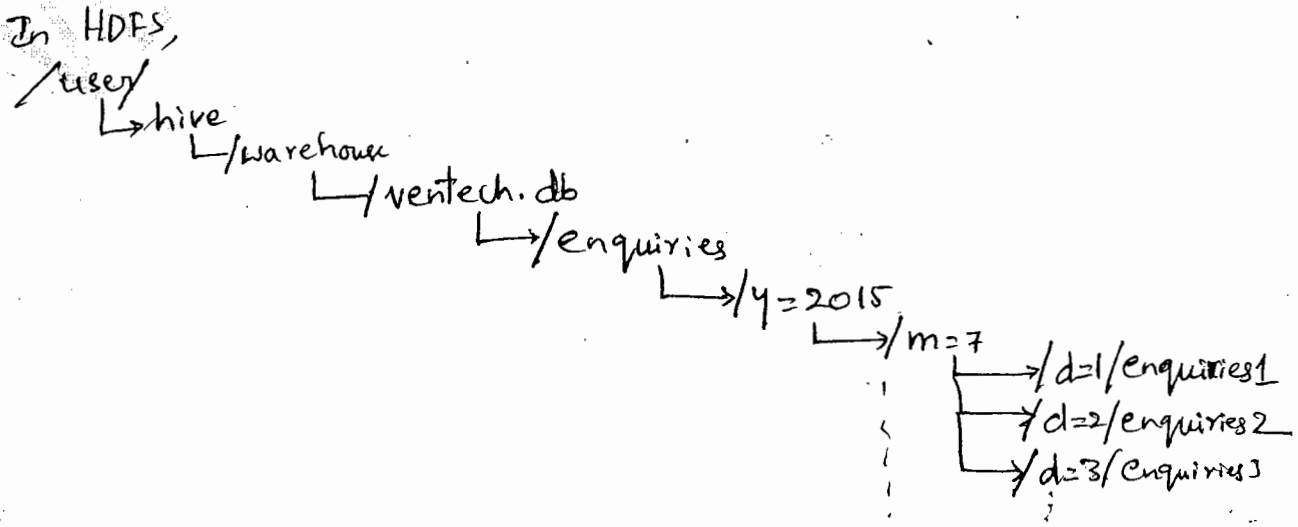
~~Problem of Partitioned Table~~

How to load data into partitions from files.

```
hive> create database ventech;
hive> use ventech;
hive> create table enquiries( name string, qualification string,
  email string, phone string, course string,
   date_of_enquiry string)
    partitioned by (y int, m int, d int)
  row format delimited fields terminated by ',';

hive> load data local inpath '/user/ventech/enquiries1'
        into table enquiries
          partition (y=2015, m=7, d=1);
hive> load data local inpath '/user/ventech/enquiries2'
        into table enquiries
          partition (y=2015, m=7, d=2);
hive> load data local inpath '/user/ventech/enquiries3'
        into table enquiries
          partition (y=2015, m=7, d=3);
```

In HDFS,



I wantto list out all enquiries from 2015-07-1 to 2015-07-10,

```
hive> select * from enquiries
      where y=2015 and m=7 and
      d<11;
```

NOW your table Enquiries is already partitioned by year, month and day.
but Sales team is intrested to access data based on course name.

```
hive> create table enquiries_on_course
      (name string, qualification string,
      email string, phone string, date_of_enquiry string)
      partitioned by (course string)
      row format delimited fields terminated by ',';
```

We need to load data from Enquiries table to Enquiries_on_Course table. but Both tables are partitioned.

How to load partioned to partitioned tables

Why we can not load it from file to Course partition.

→ Because, our enquiry files Contains all Courses. We dont have seperate files for each Course. While loading from files, we can not filter rows. So, table to table copy is only way.

Hive-43

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> From enquiries
hive> Insert overwrite table enquiries_on_course
        partition (course)
      select name, qualification, email, phone,
        date_of_enquiry, course
          from enquiries;
hive> select * from enquiries_on_course
        where course='hadoop';
```

A partitioned table can be Inner or External

Creating external partitioned table

```
hive> Create external table Ext_Part(ecode int,
        name string, sal int)
      partitioned by (dno int, sex string)
    row format delimited fields terminated by ','
    location '/user/ventech/mydata';
```

Partition tables can use custom locations.

If table has large number of partitions and partition size is large, querying data from all partitions is expensive in terms of CPU Resources and processing time.

So to restrict the user, querying data using non partitioning columns, set the following option.

hive> set hive.mapred.mode=strict;

hive> select * from mpart;

→ statement failed

because you are requesting all partitions

hive> select * from mpart

where sal>=25000;

→ statement failed

because sal is not partitioned column.

so you are requesting all partitions.

hive> select * from mpart where dno=11

→ hive reads

/user/hive/warehouse/.../mpart/d=11/s=F

/user/ ------/mpart/d=11/s=M

partitions.

In Mpart table dno, sex are partitioned Columns.

If you set,

hive> set hive.mapred.mode=nonstrict;

→ Now can request all partitions of table

→ you can use any column (partitioned / non-partitioned) in where clause.

hive-47

Up to now we have seen, ~~dif~~ advantages of partitioned tables.

Drawbacks of Partitioned Tables.

For example you are partitioning your table by city column as primary partition and dno (department number) as sub partition.

```
hive> Create table eparts (- - - - - -)
      partitioned by (city string, dno int);
hive> Set hive.exec.dynamic.partition =true;
hive> set hive.exec.dynamic.partition.mode = nonstrict;
```

Now hive has to create 10000 partitions, if have 100 unique cities and 100 unique department numbers (dnos) for each one partition one data ~~fai~~ file to be created. So 10000 files.

In this case NameNode has to maintain metadata for 10000 files.

This is greate overhead (burden) for namenode.

What if, if you are partitioning your Sales table by product_id, if you have millions of products?

→ Very very burden on Namenode for maintaining metadata.

In such case, we choose Bucketing Tables.

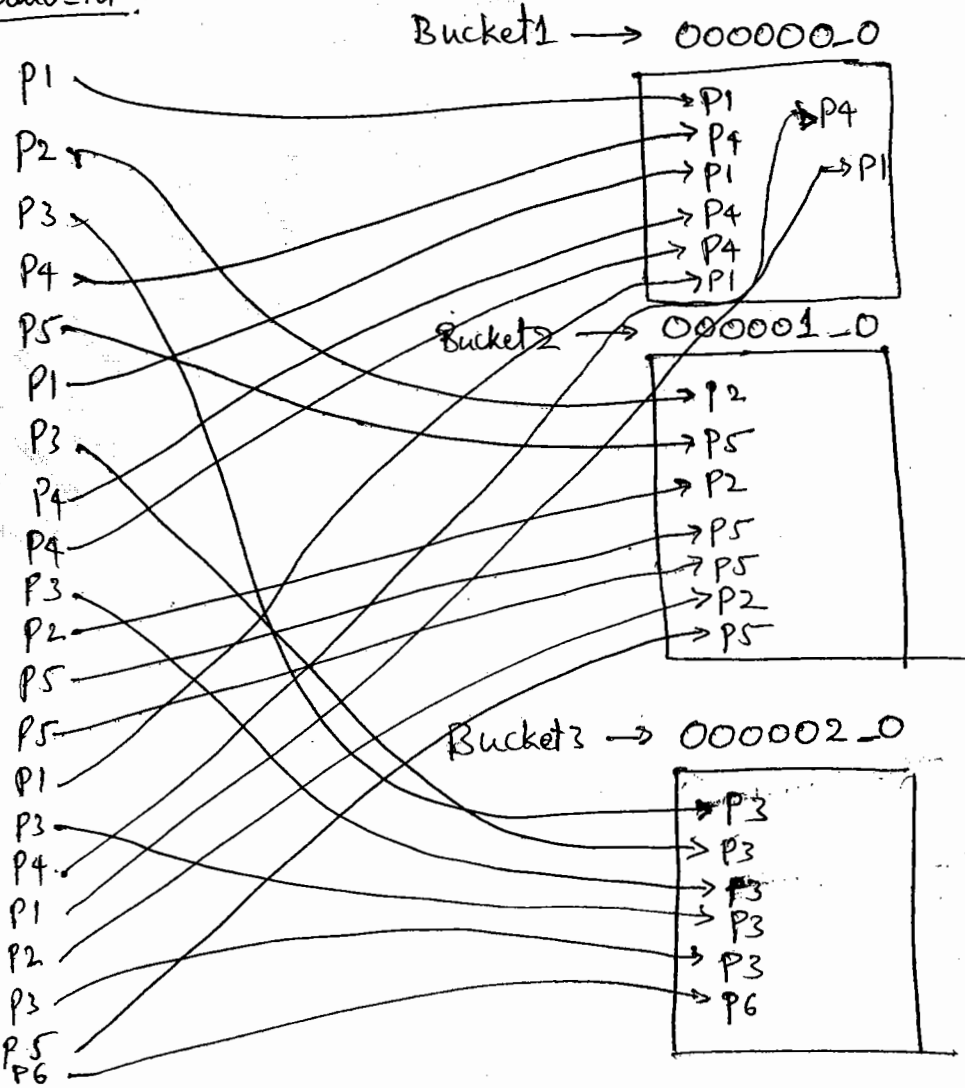
Bucketing Tables

Bucket is a file in table's directory (HDFS)

At the time of creating table,
we can choose how many buckets you want.
Each bucket may have multiple key groups.
if a group key is sent to bucket1,
all similar keys will be sent to that bucket.

ex:-

3 Buckets



observe, ~~previo per~~ from picture,

All P1, P4 product_Ids sent to 000000_0
All P2, P5 " " 000001_0
and All P3, P6 " " 000002_0

Now All data records adjusted into 3 buckets.

How to Create bucketed Tables.

```
hive> Create table emp_Buckets (ecode int,
    name string, sal int, sex string,
    dno int)
     CLUSTERED BY (dno)
     into 3 buckets;
```

loading into Buckets

```
hive> ~~Insert~~ Set hive.enforce.bucketing = true;
-- Now bucketing feature is enabled.
hive> Insert overwrite table emp_Buckets
          select * from emp;
hive> dfs -ls /user/hive/warehouse/mydb.db/emp_buckets
  →          /user - - - - - - - - /empbuckets/000000_0
             / - - - - - - - - - - - /000001_0
             / - - - - - - - - - - - /000002_0
```

Hive maintains HashTable, which key stored in which buckets.

When you request data using bucketing column, First hive understands the address of (bucketed) key by reading Hashtable, and reads data only from that bucket, with out reading from other buckets.

Hashtable.

Hash Code of key	Bucket
11	000000_0
12	000001_0
13	000002_0
14	000000_0
15	000000_0
16	000002_0
17	000003_0
18	000003_0


```
hive> select * from emp_backets;
```

→ hive reads all buckets;

```
hive> select * from emp_buckets
      where dno=16;
```

Now hive Reads data from only 000002_0 bucket.

you can create bucketed tables using multiple columns.

example.

```
hive> Create table bucks_tab(ecode int,
      name string, sal int, sex string,
      dno int)
      Clustered by (dno, sex)
      into 3 buckets;

hive> set hive.enforce.bucketing=true;

hive> Insert overwrite table bucks_tab
          select * from Emp;
```

now Combination of dno, sex act as a Composite key.

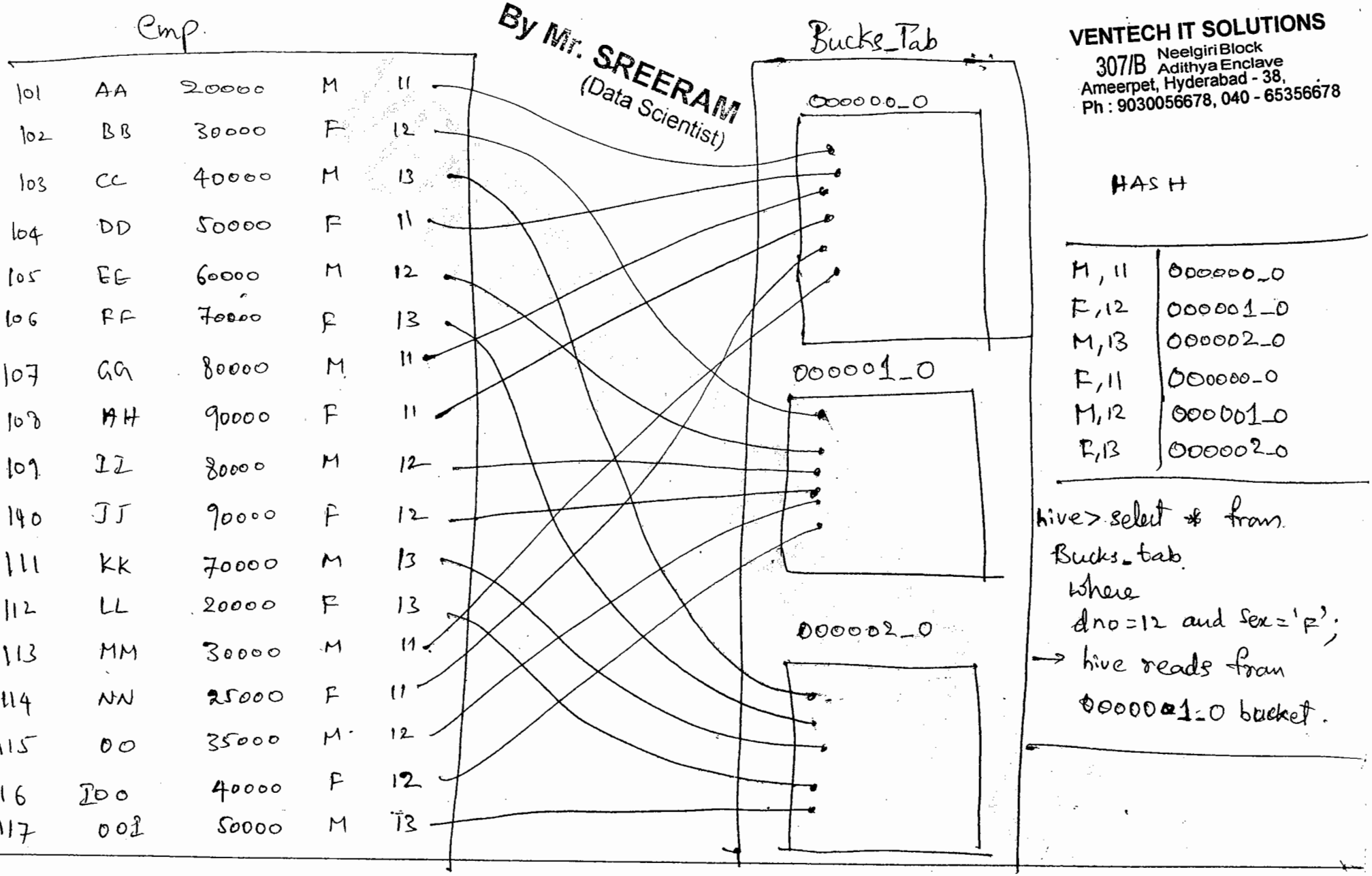
if 11, F (dno, sex) is entered into bucket1, all remaining 11, F records will be sent to bucket1.

(Lets see the figure.. in next page)

```
hive> selet * from bucks_tab
          where dno=11 and sex='F';
```

hive reads from bucket1 (000000_0) file.

VENTECH IT SOLUTIONS
307/B Neelgiri Block
Adithya Enclave
Ameerpet, Hyderabad - 38,
Ph : 9030056678, 040 - 65356678



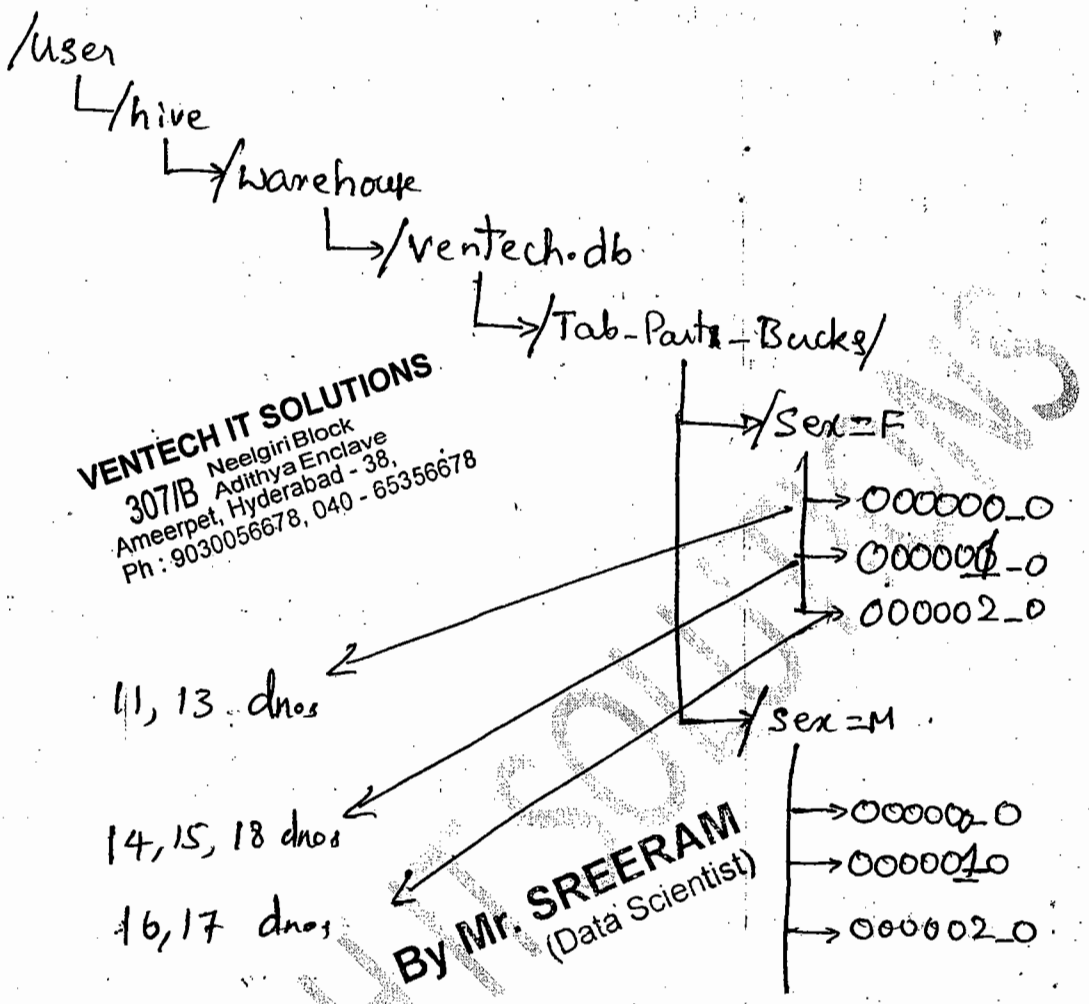
HASH

M, 11	000000_0
F, 12	000001_0
M, 13	000002_0
F, 11	000000_0
M, 12	000001_0
F, 13	000002_0

```
hive> select * from
  Bucks_tab
  where
  dno=12 and sex='F';
```

→ hive reads from 000001_0 bucket.

BY Mr SREERAM



hive> select * from Tab_Part_bucks
where dno in (14,18) and sex='F';

→ Hive reads data from
/user/hive/warehouse/ventech.db/Tab_part_Bucks/sex=F/000001_0
(file)

hive> select * from Tab_Part_bucks
where sex='F';

→ hive reads data from all buckets of sex=F partition (directly).

We can create tables with combination of partitions and Bucketing feature.

examples.

```
hive> use ventech;
hive> Create table Tab_Part_Bucks (ecode int,
      name string, sal int, city string,
      dno int)
      partitioned by (sex string)
      clustered by (dno int) Into 3 Buckets
      row format delimited
          fields terminated by ',';

hive> set hive.enforce.bucketing = true;
hive> set hive.exec.dynamic.partition = true;
hive> set hive.exec.dynamic.partition.mode = nonstrict;
hive> Insert overwrite table Tab_part_Bucks
          partition (sex)
      select ecode, name, sal, city, dno, sex
          from emp;
```

In HDFS,

→